Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

PROCESSO DE KDD PARA AUXÍLIO À RECONFIGURAÇÃO DE AMBIENTES VIRTUALIZADOS

Ana Trindade Winck

Dissertação de Mestrado

Orientador: Prof. Dr. Duncan Dubugras Alcoba Ruiz

Porto Alegre
2007

Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

PROCESSO DE KDD PARA AUXÍLIO À RECONFIGURAÇÃO DE AMBIENTES VIRTUALIZADOS

Ana Trindade Winck

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Duncan Dubugras Alcoba Ruiz

Porto Alegre
2007

Dados Internacionais de Catalogação na Publicação (CIP)

W761p Winck, Ana Trindade
Processo de KDD para auxílio à reconfiguração de ambientes virtualizados / Ana Trindade Winck. – Porto Alegre, 2007.
78 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Duncan Dubugras Alcoba Ruiz.

Ficha Catalográfica elaborada pelo
Setor de Processamento Técnico da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Processo de KDD para Auxílio à Reconfiguração de Ambientes Virtualizados**", apresentada por Ana Trindade Winck, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 20/12/07 pela Comissão Examinadora:

Prof. Dr. Duncan Dubugras Alcoba Ruiz – Orientador — PPGCC/PUCRS

Prof. Dr. Avelino Francisco Zorzo – PPGCC/PUCRS

Prof. Dr. Leandro Krug Wives – UFRGS

Homologada em 22/01/08, conforme Ata No. 003 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

À minha família

AGRADECIMENTOS

Agradeço ao meu marido, Paulo, por todo o seu amor e por todo o incentivo durante todos os momentos em que estivemos juntos. Aos meus pais, Irani e Rosecler, por sempre terem confiado e me apoiado em todas as escolhas, e por serem os responsáveis por me fazer ser o que sou hoje. À minha vó Noely e à minha dinda Rosângela, que sempre me acompanharam de perto. Essas cinco pessoas, tenho certeza, sempre torceram para que tudo desse certo. E deu! Amo vocês!

Agradeço o meu orientador, Duncan, pela oportunidade e por ter confiado em mim e no meu trabalho. Aos colegas do GPIN, em especial a toda a galera de 2006 e, principalmente, às minhas grandes amigas Tita, Fer e Pati!

Ao professor Avelino, pela oportunidade em participar do projeto Xen. Aos colegas do projeto, pela cooperação. E à HP Brasil P&D, por ter financiado meus estudos.

Aos demais colegas e amigos da PUC, por tudo o que passamos juntos. Aliás, só vocês sabem, de fato, o que significam esses dois anos!

Aos que não nomeei, mas que participaram, de perto ou de longe, dessa trajetória. Enfim, a todos os que colaboraram para que isso tudo acontecesse! Muito obrigada!

RESUMO

Xen é um paravirtualizador que permite a execução simultânea de diversas máquinas virtuais (VM), cada uma com seu próprio sistema operacional. O consumo dessas VMs se dá em diferentes níveis de recursos. Com o objetivo de melhorar a *performance* do Xen, é interessante verificar qual a melhor alocação de recursos para uma dada máquina Xen, quando várias VMs são executadas, e quais são os respectivos parâmetros. Para auxiliar a eventual reconfiguração de parâmetros, este trabalho propõe um processo completo de descoberta de conhecimento em banco de dados (processo de KDD) para capturar dados de desempenho das VMs, organizá-los em um modelo analítico e aplicar técnicas de mineração para sugerir novos parâmetros. Inicialmente são obtidos dados de desempenho de cada VM, onde a estratégia empregada é a execução de *benchmarks* sobre cada sistema operacional. Esses dados são armazenados em um *data warehouse* propriamente modelado para armazenar registros de captura de métricas de *benchmarks*. Os dados armazenados são convenientemente preparados para serem utilizados por algoritmos de mineração de dados. Os modelos preditivos gerados podem, então, ser enriquecidos com instruções em alto nível de reconfigurações. Tais modelos buscam sugerir, dada uma configuração vigente, qual o melhor conjunto de parâmetros de configuração para modificar o ambiente, e alcançar um ganho global de desempenho. O processo proposto foi implementado e testado com um conjunto significativo de execuções de *benchmarks*, o que mostrou a qualidade e abrangência da solução.

Palavras-Chave: Xen, KDD, Data Warehouse, Preparação de Dados, Mineração de Dados.

ABSTRACT

Xen is a paravirtualizer that allows the simultaneous execution of several virtual machines (VM), each with its own operating system. Inputs for these VMs occur at different resource levels. When the aim is to improve Xen performance, it is interesting to assess the best resource allocation for a given Xen machine when different VMs are executed and the respective parameters adopted. This study puts forward a complete process of knowledge discovering in databases (KDD process). The aim of the process is to (i) capture VM development data, (ii) organize these data as an analytical model, and (iii) implement data mining techniques to suggest new parameters. First, VM development data are obtained by benchmarking each operating system. These data are stored in a data warehouse specially modeled so as to store capture records of benchmark metrics. The data stored are conveniently prepared to be used by data mining algorithms. The predictive models generated are enriched with high-level reconfiguration instructions. These models aim at suggesting the best set of configuration parameters to modify the environment and reach an overall gain in performance, for a given configuration in use. The process proposed was initially implemented and tested in a significant set of benchmarking executions, proving the quality and range of the solution.

Keywords: Xen, KDD, Data Warehouse, Data Preparation, Data Mining.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ARFF	*Attribute-Relation File Format*
BM	*Benchmark*
CAP	Topo
CPU	*Central Processor Unit*
DSA	*Data Staging Area*
DW	*Data Warehouse*
E/S	Dispositivos de Entrada e Saída
ETC	Extração, Transformação e Carga
FN	Falso Negativo
FP	Falso Positivo
HW	*Hardware*
KBPS	*Kbytes per Second*
KDD	*Knowledge Discovery in Databases*
LPM	*Loops per Minute*
LPS	*Loops per Second*
OLAP	*On-Line Analytical Processing*
PV	Paravirtualizador
SLA	*Service Level Agreement*
SLO	*Service Level Objective*
SO	Sistemas Operacionais
VM	*Virtual Machine*
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

SUMÁRIO

1. INTRODUÇÃO

A virtualização de sistemas operacionais (SO) é uma prática que tem sido explorada para permitir a execução simultânea de múltiplos SOs em uma única máquina física [MER06]. Xen [BAR03] é um paravirtualizador que oferece tal recurso. A arquitetura deste é representada como uma camada de abstração sobre o hardware, a qual permite a execução de diversas máquinas virtuais (VM) paralelas, cada uma com seu próprio SO. Sua empregabilidade, motivada pela economia de recursos físicos, ganha destaque em *data centers* [CHE07] que, em função de sua demanda no fornecimento de serviços, podem compartilhar, para vários clientes, uma mesma infra-estrutura computacional.

Com o objetivo de propor melhorias no que se refere à *performance* do Xen, uma importante questão a ser endereçada é: qual a melhor alocação de recursos para uma máquina Xen quando várias VMs estão sendo executadas? Para tanto, torna-se fundamental saber com precisão o desempenho dessas VMs. Uma boa prática para avaliação de desempenho de sistemas operacionais é através da utilização de *benchmarks*, os quais buscam extrair métricas e resultados relacionados ao desempenho das mesmas. Com essas métricas, espera-se poder predizer características de desempenho e, a partir dessas, sugerir novos parâmetros de configuração, para que seja possível alcançar um ganho de desempenho.

Este trabalho propõe um processo de descoberta de conhecimento em banco de dados (processo de KDD) para contribuir na avaliação de desempenho e na sugestão de parâmetros para a reconfiguração do Xen. Para tanto, dados de execuções de *benchmarks* sobre os sistemas operacionais virtualizados são compilados e organizados em um *data warehouse* (DW) propriamente modelado. Esses dados, após uma preparação adequada, são utilizados por técnicas de mineração de dados que produzem modelos preditivos que indicam, dada uma configuração vigente de uma máquina Xen, quais parâmetros de configuração merecem ser alterados para atingir um ganho de desempenho. O processo proposto é testado com um número significativo de execuções de *benchmarks*, para mostrar a qualidade e abrangência da solução. Esta pesquisa está inserida em um projeto de parceria, onde os esforços em reconfigurar o Xen são endereçados por duas pesquisas complementares.

Este trabalho está estruturado conforme segue. O Capítulo 2 apresenta uma revisão sobre o referencial teórico, abordando os contextos de virtualização e de processo de KDD. No Capítulo 3 é apresentado o cenário desta pesquisa, caracterizando o problema e relatando a

solução para o mesmo, onde é apresentado o processo de KDD construído. O Capítulo 4 apresenta a etapa de *data warehousing* do processo de KDD, apresentando como os dados são capturados e armazenados em um ambiente de DW desenvolvido. No Capítulo 5 é mostrado como técnicas de mineração de dados podem fazer uso dos dados contidos no DW para sugerir parâmetros de reconfiguração do Xen. O Capítulo 6 apresenta os trabalhos relacionados a esta pesquisa. No capítulo 7 são apresentadas as conclusões e trabalhos futuros e, em seguida, as referências bibliográficas.

2. REFERENCIAL TEÓRICO

Para contextualizar a pesquisa, este capítulo apresenta uma revisão bibliográfica sobre a virtualização de sistemas operacionais e sobre o processo de KDD. Primeiramente é apresentado o paravirtualizador Xen, destacando suas características e mostrando como o desempenho do mesmo pode ser avaliado e melhorado. Em seguida é relatado o processo de KDD, explorando cada uma de suas etapas.

2.1. Virtualização de Sistemas Operacionais

Virtualização é uma técnica que permite que múltiplos sistemas operacionais possam coexistir em uma única máquina física, cada um executando em uma máquina virtual [MER06]. Como extensão à virtualização, existe a técnica denominada paravirtualização [MEN05]. Esta última caracteriza-se por ser uma camada de abstração existente entre o hardware e as VMs (onde cada VM tem seu próprio sistema operacional). Esta camada é responsável por permitir o gerenciamento de instruções privilegiadas (também denominada de Domínio 0) dos sistemas operacionais em relação ao hardware. Xen [BAR03] é um paravirtualizador que permite a execução simultânea de múltiplos sistemas operacionais com o compartilhamento de um mesmo hardware. A Figura 1 ilustra a arquitetura do Xen, onde observa-se que o mesmo é uma camada acima do hardware, no caso suportando 4 VMs. Vale ressaltar que o Xen não se limita à utilização de apenas 4 VMs, [BAR03] relata experimentos efetuados com até 100 VMs.

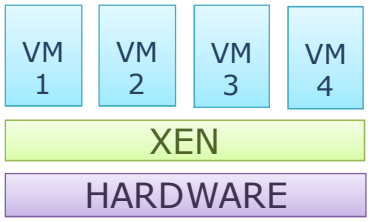

Figura 1 – Arquitetura do Xen. Adaptado de [BAR03]

A utilização do Xen é motivada pela potencial economia de recursos físicos que o mesmo pode proporcionar. Essa abordagem torna-se vantajosa quando os recursos computacionais disponíveis para o ambiente virtualizado são bem aproveitados por cada VM, contribuindo para que as mesmas apresentem um desempenho mais satisfatório.

Sabendo-se que cada VM suporta um conjunto distinto de aplicações, e que cada aplicação consome diferentes níveis de recursos computacionais, é interessante que esses sejam adequadamente distribuídos no ambiente. Para tanto, alguns critérios de medições devem ser estabelecidos. No caso do Xen, [BAR03], [CLA05] e [ROS05] sugerem que tais critérios sejam focalizados em desempenho da CPU, de gerenciamento de memória e de dispositivos de E/S de cada sistema operacional virtualizado. Embora esses níveis de desempenho possam ser fornecidos pelo próprio sistema operacional, existem outras abordagens para extrair métricas de desempenho, como é o caso de *benchmarks* para sistemas operacionais.

2.1.1. Análise de Desempenho por Benchmarks

Para identificar, antecipadamente, quais níveis de consumo cada sistema operacional é capaz de consumir, sem que haja necessidade de executar aplicações reais, é possível configurar o ambiente disponibilizando diferentes limites de consumo para cada VM. Como os *benchmarks* buscam estressar a capacidade operacional de cada sistema, ao executar um *benchmark* para cada sistema operacional virtualizado, é esperado que o limite de consumo atribuído para cada VM seja totalmente utilizado.

Existem distintos *benchmarks* para medir desempenho de sistemas operacionais, dentre os quais pode-se destacar Unixbench [UNI07], LmBench [MCV96] e Iozone [IOZ07]. Cada *benchmark* realiza medições em diferentes aspectos. Os dois primeiros, por exemplo, focalizam suas medições em CPU e memória, e o último em E/S. A forma padrão com que esses *benchmarks* apresentam seus resultados é via relatório tabular, contendo métricas de desempenho e os respectivos valores encontrados. Cada *benchmark* faz uso de critérios próprios para apresentação e disposição dos resultados.

A Figura 2 ilustra um exemplo de resultado de uma execução do *benchmark* Unixbench, a qual mostra o resultado de desempenho de uma VM em um ambiente com quatro VMs paralelas. Seu formato de apresentação é estruturado conforme segue e indicado na Figura 2: (a) as cinco primeiras linhas apresentam informações quanto às configurações do *benchmark* e do ambiente sobre o qual este foi executado; (b) em seguida são listadas diferentes métricas, os valores encontrados, suas respectivas unidades de medida e a forma de medição das mesmas; (c) logo abaixo é apresentado um resumo das informações, de onde são listadas as métricas mais relevantes contendo: o nome da métrica (*test*), o valor ideal a ser

reportado (*baseline*) o valor encontrado (*result*), e um índice que representa o percentual de aproveitamento do valor encontrado em função do valor ideal (*index*) (vale ressaltar, que tanto a classificação de relevância, como os valores apresentados são efetuados de acordo com algum critério do próprio *benchmark*); (d) por fim, é apresentado o desempenho total do sistema, através do somatório (*sum*) e da média (*average*) dessas métricas.

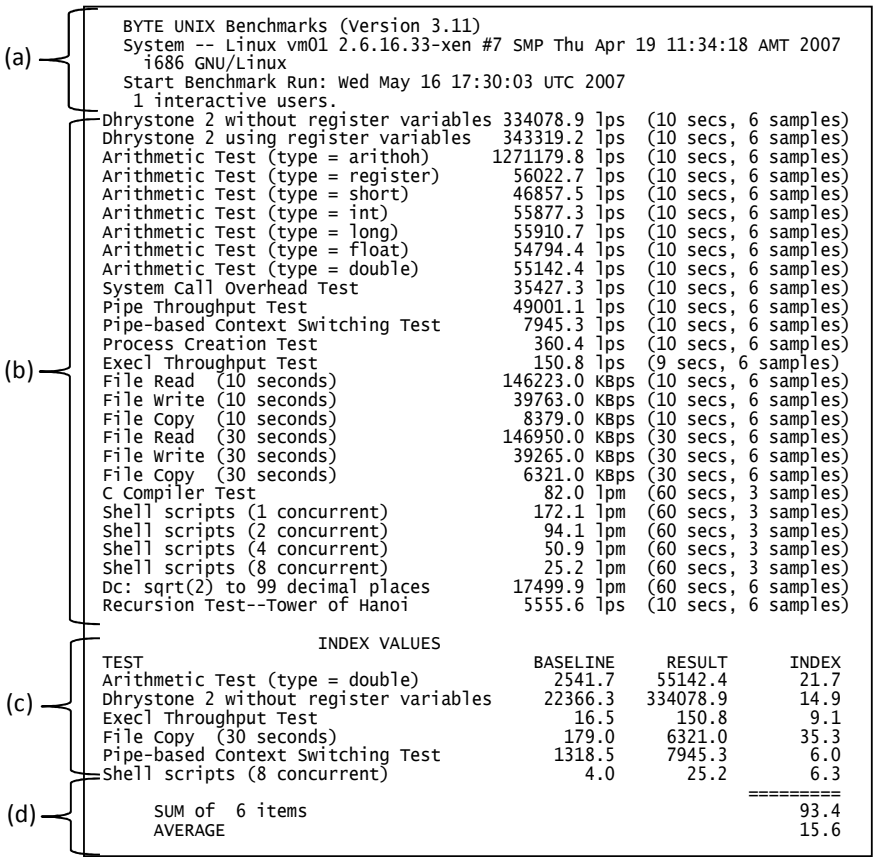

Figura 2 – Exemplo de resultado do Unixbench

Para cada VM analisada é produzido um relatório similar ao apresentado na Figura 2, cada um com seus respectivos valores encontrados para cada métrica. Os resultados do *benchmark* podem ser analisados sobre diferentes óticas. Como cada métrica apresenta desempenho em diferentes aspectos, pode-se selecionar apenas aquelas que atendam aos critérios previamente estabelecidos. Por outro lado, também é possível analisar a *performance* total do sistema operacional reportado: no caso do *benchmark* apresentado no exemplo, pelo valor da média (*avarege*, Figura 2d). De posse dos resultados individuais de cada sistema operacional, é possível obter o desempenho global do ambiente virtualizado, através da soma dos resultados de desempenho de cada VM.

Analisar o conjunto de resultados e, a partir deles, inferir sobre a adequada distribuição das VMs no ambiente virtualizado, não é uma tarefa trivial de ser efetuada manualmente. Nesse sentido, é conveniente fazer uso de algum suporte computacional que dê um adequado apoio para a análise de desempenho, apresentando sugestões para distribuição de recursos nas VMs.

2.2. Processo de KDD

O processo de descoberta de conhecimento em base de dados (KDD – *Knowledge Discovery in Databases*) [FAY96] [HAN01] [LIR07] é uma seqüência de etapas interativas e iterativas que endereçam a tomada de decisão, especialmente quando da existência de um grande volume de dados. Conforme ilustrado pela Figura 3, o processo de KDD propõe que, a partir de uma coleção distinta de dados (a), seja constituído um *data warehouse* (c) para que, com os dados dele, sejam utilizadas técnicas de mineração de dados (e) para produzir conhecimento útil ao objetivo para o qual esse processo esteja sendo endereçado (f). A qualidade do conhecimento útil fornecido depende da qualidade dos dados utilizados. Para tanto, as etapas de transformação (b) e preparação (d) são primordiais para garantir essa qualidade. Estas etapas, juntas, consomem em torno de 85% [LIR07] de todo o processo de KDD. A transformação visa coletar dados que sejam relevantes ao problema e adequadamente transformá-los para serem inseridos no ambiente de DW. A preparação tem por objetivo preparar os dados já acomodados no DW de modo que, ao executar um algoritmo de mineração sobre esses dados, os mesmos sejam especialmente úteis para atender o problema endereçado.

Embora cada etapa que compõe o processo de KDD seja suficientemente abrangente para ser tratada individualmente, existe uma forte relação de dependência entre elas. Assim, para que seja efetuada uma correta transformação dos dados, é necessário ter um ambiente de DW corretamente modelado. Da mesma forma, para que os dados sejam apropriadamente preparados, é preciso ter como alvo um algoritmo de mineração selecionado para atender ao problema e alcançar o objetivo esperado.

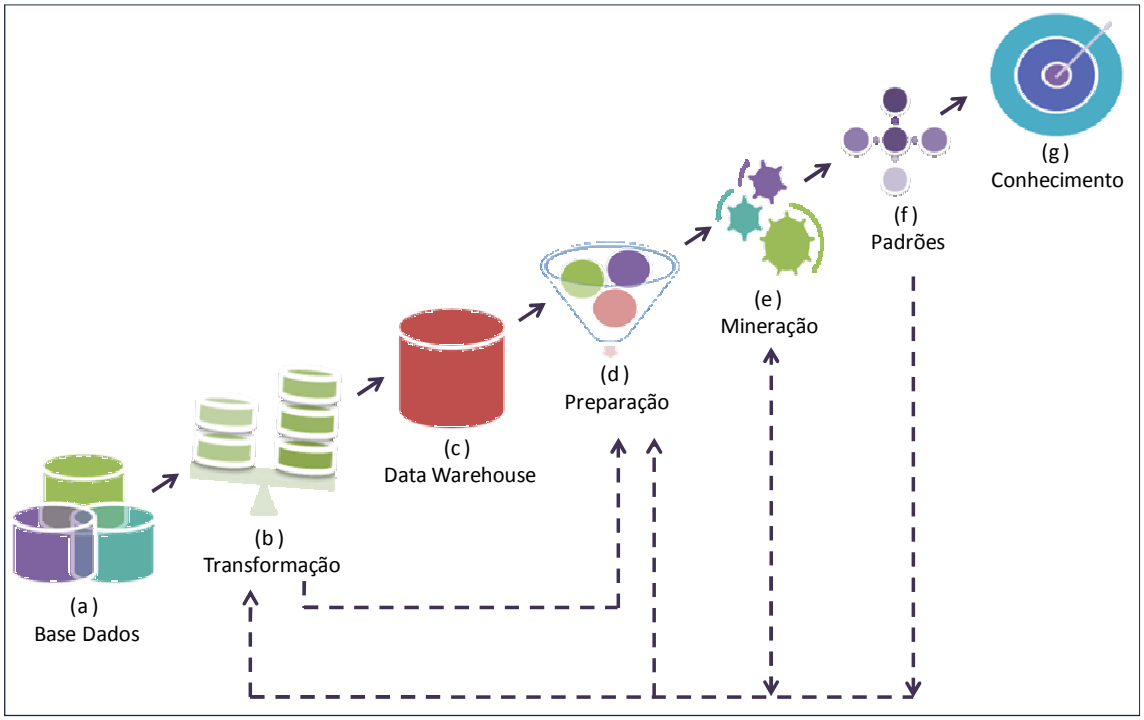

Figura 3 – Processo de KDD. Adaptado de [HAN01]

2.2.1. Data Warehouse

Em um DW são inseridas instâncias provenientes de distintas bases de dados, buscando manter um histórico dos registros. Os dados contidos no DW devem estar organizados em contextos que auxiliem à tomada de decisão estratégica [HAN01].

DW, por sua essência, são construídos de forma a satisfazer sua estrutura multidimensional [KIM98]. Um modelo analítico, que representa essa estrutura multidimensional, possui dois tipos de tabelas: fato e dimensão. Entende-se por dimensão as perspectivas de uma base de dados que possam gerar registros referentes às características do problema modelado. Essas são, tipicamente, organizadas em torno de um termo central, ou tabela fato, a qual contém os atributos chave das dimensões e atributos que representem valores relevantes ao problema. Existem três tipos principais de modelo analítico:

- **Modelo Estrela –** o modelo estrela caracteriza-se, basicamente, por conter uma única tabela fato e várias dimensões;

- **Modelo Floco de Neve –** o modelo floco de neve se difere do modelo estrela por admitir que as tabelas dimensões formem hierarquias, com relacionamentos entre si; e
- **Modelo Constelação de Fatos –** de uma forma mais abrangente, no modelo constelação de fatos é possível dispor de mais de uma tabela fato, as quais possuem suas próprias dimensões, e tais dimensões, também podem formar hierarquias, com relacionamentos entre si.

O que determina a correta escolha do modelo analítico a ser modelado para o DW é tanto as características do problema, como as características de suas bases de dados. Embora o modelo mais usual seja o estrela, determinadas características do problema podem demandar a aplicação de outros tipos de modelo. No que tange ao modelo floco de neve, [KIM98] sustenta que este modelo deve ser utilizado com cautela, uma vez que a hierarquização aplicada pode não ser clara para o usuário do modelo.

2.2.2. Extração, Transformação e Carga

Um dos processos mais importantes para a construção de um DW é a etapa de extração, transformação e carga (ETC) [KIM98] das fontes de dados para o DW. Esta etapa busca aumentar a qualidade dos dados a serem inseridos no repositório. No processo de KDD, a ETC está representada pela etapa de transformação (Figura 3b).

A extração é a primeira fase do processo de ETC, a qual visa percorrer as distintas bases de dados e capturar apenas os dados significantes ao domínio. Na fase de transformação, os dados capturados podem sofrer algum tipo de tratamento, ajustando possíveis conflitos de tipificação e eliminando ruídos ou possíveis conteúdos irrelevantes ao DW. Por fim, é feita a devida carga dos dados já transformados no ambiente de DW, obedecendo à estrutura definida no modelo analítico.

Para apoiar a etapa de ETC, [KIM98] defende a criação de um DSA (*Data Staging Area*), uma área temporária à disposição dos dados extraídos e que estão passando pelo processo de transformação. Nessa área os dados devem ser mantidos antes de serem efetivamente carregados no DW.

O processo de construção do DW juntamente com a efetivação da etapa de ETC é chamado de processo de *data warehousing* [BEC06]. Com esse, os dados são adequadamente

inseridos no DW construído que, portanto, já admite a manipulação por ferramentas OLAP para pivoteamento de dados, *roll-up*, *drill-down* e *slice&dice*, operações essas típicas de ambientes de processamento analítico de dados. Além dessa abordagem analítica, os dados podem ser utilizados por algoritmos de mineração de dados.

2.2.3. Mineração de Dados

A mineração de dados é a etapa do processo de KDD que converte dados brutos em informação. A mineração de dados divide-se em diferentes técnicas: descritivas e preditivas [TAN06]. A primeira sumariza relações entre dados, tendo como objetivo aumentar o entendimento a respeito desses. A segunda tem o objetivo de apontar conclusões quanto aos dados analisados, prevendo valores para um dado atributo de interesse.

Técnicas preditivas de mineração de dados, especialmente tarefas de classificação, buscam construir modelos preditivos que apresentem a melhor combinação de relacionamentos entre um conjunto de atributos, denominados atributos explanatórios, e um dado atributo de interesse, denominado atributo preditivo [TAN06]. A classificação [HAN01] descreve e distingue o atributo preditivo, tal que os modelos resultantes possam ser utilizados para predizer a classe cujos atributos explanatórios pertencem. A fim de que esse modelo seja produzido, o classificador passa por uma etapa de treinamento. O treinamento do classificador é feito a partir de uma seleção de dados denominada conjunto de treino. Este consiste em registros cujo atributo preditivo deve ser previamente fornecido para que seja constituído um modelo de classificação. Após o treinamento, é aplicado o conjunto de teste, o qual consiste em uma seqüência de registros cujos atributos são o mesmo do conjunto de treino, mas o valor atributo preditivo não é conhecido. É a partir do treinamento que o classificador produz modelos preditivos para o conjunto de teste.

Existem diferentes técnicas que podem representar um classificador, como árvores de decisão, redes neurais, redes bayesianas, entre outros. Uma árvore de decisão, por exemplo, é um grafo cujos nodos internos são compostos pelos atributos explanatórios, e os nodos folha por valores para o atributo preditivo. A idéia é dividir os atributos explanatórios, baseado em testes de condição, até que o nodo folha seja atingido indicando, assim, qual o valor do atributo preditivo. O C4.5 [QUI96] é um popular algoritmo que implementa árvores de decisão. O ambiente Weka [WEK07] chama de J48 a implementação do C4.5.

2.2.3.1. Formatos de Entrada para o Algoritmo de Mineração

Embora ferramentas de mineração de dados, como o Weka, sejam capazes de se comunicar diretamente com um DW, pode não ser conveniente utilizar os dados em sua forma bruta, assim como pode não ser conveniente que todos os atributos sejam utilizados. Para que um algoritmo de mineração possa ler os dados a serem minerados, os mesmos devem ser convertidos em algum formato que o mesmo seja capaz de interpretar. No caso do Weka, o formato preferencial para interpretação dos atributos a serem minerados é a partir de um arquivo de extensão arff (*attribute-relation file format*), obedecendo a formatação imposta pelo mesmo.

A Figura 4 exemplifica um arquivo no formato arff. Esse tipo de arquivo é composto de atributos e instâncias para estes atributos. A primeira entrada deste arquivo (a) determina o nome do mesmo. Em seguida (b), são declarados todos os atributos que compõe o arquivo. Estes atributos podem ser de dois tipos: contínuos ou categóricos. Quando contínuo, basta informar o tipo de valor aceito (c): no exemplo, real. Quando categóricos, é necessário informar todas as entradas válidas para cada atributo (d): {a, b, c} para Atributo2 e {0, 1} para Atributo3. Após a declaração, são inseridas as instâncias (e) do arquivo, obedecendo à ordem declarada dos atributos e separando os mesmos por vírgula. Todas as entradas precedidas do sinal @ (arroba) são mandatórias para a correta formatação do arquivo.

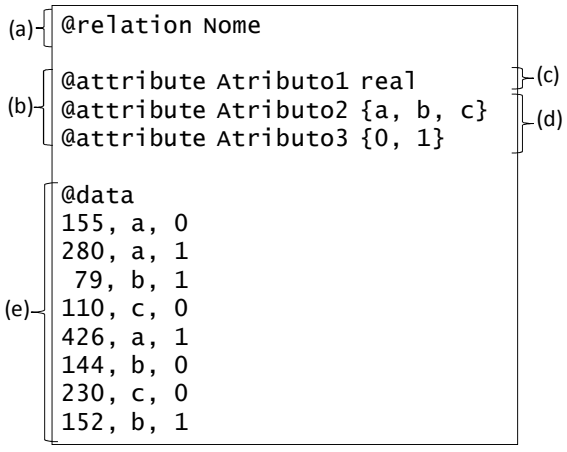

Figura 4 – Exemplo de arquivo no formato arff

2.2.3.2. Qualidade dos resultados

Para utilizar o algoritmo J48, a ferramenta Weka permite que o conjunto de treino e o conjunto de teste sejam fornecidos em arquivos individuais. Entretanto, a opção padrão sugere a utilização do método de validação cruzada (*cross-validation*). Esse método [WIT05], ao ler um arquivo arff, divide o mesmo em conjunto de treino e conjunto de teste. É sugerido, como padrão, que o mesmo seja dividido em dez partes, sendo que uma dessas partes é escolhida, aleatoriamente, para ser usada como conjunto de treino, enquanto as outras nove partes são utilizadas como conjunto de teste. É possível determinar em quantas partes deseja-se dividir o arquivo. Entretanto, em [WIT05] é relatado que diversos experimentos mostraram que a divisão em 10 partes apresenta os resultados mais satisfatórios.

A qualidade do resultado fornecido pelo algoritmo pode ser verificada pela matriz de confusão (*confusion matrix*) fornecida. A partir do número total de instâncias contidas no arquivo, essa matriz mostra, para cada valor do atributo preditivo, o número de instâncias que foram classificadas corretamente e o número de instâncias que foram classificadas erroneamente. A Tabela 1 mostra como essa matriz é apresentada onde, no caso de o atributo preditivo conter apenas dois valores (0 e 1), têm-se:

- O número de instâncias classificadas como verdadeiros positivos (VP);
- O número de instâncias classificadas como verdadeiros negativos (VN);
- O número de instâncias classificadas como falsos positivos (FP); e
- O número de instâncias classificadas como falsos negativos (FN).

Tabela 1 – Matriz de confusão. Adaptado de [WIT05]

Atributo Preditivo		**Valor Predito**	
		1	**0**
Valor Real	**1**	VP	FN
	0	FP	VN

Com os resultados da matriz de confusão é possível determinar a acurácia do modelo, ou seja: quão satisfatória foi a classificação. A acurácia é calculada por:

$$Acurácia = \frac{Número\ de\ predições\ corretas}{Número\ total\ de\ predições} = \frac{VP+VN}{VP+VN+FP+FN}$$

2.2.4. Preparação de Dados

A forma de apresentação dos dados contidos no arquivo a ser interpretado pelo algoritmo de mineração pode ser fundamental para que este algoritmo retorne resultados com melhor acurácia. Para tanto, é sugerido [TAN06] [HAN01] [FAY96] [LIR07] que os dados a serem minerados passem por uma conveniente etapa de preparação para a mineração (a qual não deve ser confundida com o processo de ETC). Esta busca trabalhar com os dados que já estão contidos no DW e adequá-los para a mineração.

Mesmo que a preparação de dados seja bastante trabalhosa, demandando bastante tempo, este esforço deve ser endereçado por contribuir fortemente para o grau de confiabilidade dos resultados obtidos pelos diferentes algoritmos de mineração empregados. Assim, a conveniente preparação dos dados para a mineração é um ponto importante a ser endereçado. Tan [TAN06] propõe uma série de etapas que auxiliam tal preparação:

- **Agregação –** essa etapa busca sumarizar os dados em diferentes perspectivas como, por exemplo, combinando dois ou mais objetos em um único objeto, reduzindo o escopo a ser minerado;
- **Suavização –** a suavização dos dados visa eliminar possíveis ruídos que possam estar presentes no conjunto de dados analisado;
- **Redução de Dimensionalidade –** a redução de dimensionalidade busca sumarizar em um volume menor de dados, atributos que possuem uma grande quantidade de valores distintos;
- **Seleção de Atributos –** essa técnica visa eliminar atributos (colunas) que possam ser redundantes e irrelevantes ao objetivo da mineração, onde o subconjunto selecionado seja tão representativo quanto seriam os dados originais;
- **Criação de Atributos –** essa técnica permite que, baseado em valores de atributos já existentes, seja possível criar outros atributos que, em um escopo menor, possam melhor representá-los;
- **Discretização –** a discretização permite transformar atributos contínuos em atributos discretos. Além de alguns algoritmos de mineração não aceitarem valores contínuos (no caso da classificação, é essencial que, pelo menos, o

atributo preditivo seja discreto), alguns valores contínuos podem estabelecer alguma relação de ordem entre si que sejam desnecessárias ou até perturbadoras; e

- **Transformação de Atributos –** a transformação de atributos busca aplicar alguma regra que seja inferida sobre os valores de um dado atributo como, por exemplo, normalizar uma escala de valores.

2.3. Considerações do Capítulo

Este capítulo apresentou uma revisão bibliográfica dos assuntos endereçados nesta pesquisa. Foi discutido o conceito de virtualização e apresentado o Xen, um paravirtualizador que permite a execução simultânea de distintas VMs, cada uma com seu próprio sistema operacional, oferecendo acesso às instruções privilegiadas do hardware. Para tanto, foi mostrado como a utilização de *benchmarks* pode ser útil para avaliar o desempenho dessas VMs.

Foi discutido, também, o processo de KDD, o qual endereça a tomada de decisão. Foi enfatizado que a qualidade dos resultados obtidos com o processo de KDD está diretamente relacionada à qualidade dos dados que compõem o DW e à qualidade dos dados a serem minerados. Assim, entende-se que essas são etapas que merecem especial atenção. No capítulo seguinte é relatado como um processo de KDD pode auxiliar na reconfiguração de ambientes virtualizados.

3. DESCRIÇÃO DO CENÁRIO

A Figura 5 apresenta uma típica atuação do Xen, onde se tem, no exemplo, quatro VMs (1, 2, 3, 4), sendo que cada uma dessas VMs têm níveis diferentes de consumo de recursos (ex: CPU e memória), ilustrado pela altura de cada VM. As linhas tracejadas representam a disponibilidade de recursos para o ambiente. Parte-se do pressuposto que, ao inicializar uma máquina Xen, os recursos disponíveis sejam igualmente alocados para todas as VMs. Essa alocação está representada pela Figura 5a, onde percebe-se que a VM 2 está subutilizando os recursos a ela disponíveis, enquanto que a necessidade de consumo da VM 3 é maior do que o limite à sua disposição, não sendo possível atender a toda sua demanda. Para melhorar a *performance* global do Xen, espera-se que os recursos possam ser adequadamente redistribuídos. Este modelo ideal está representado na Figura 5b, onde é possível notar, pela linha tracejada, que os recursos computacionais foram distribuídos proporcionalmente à demanda de cada VM, sem excesso ou escassez, respeitando a disponibilidade global do ambiente.

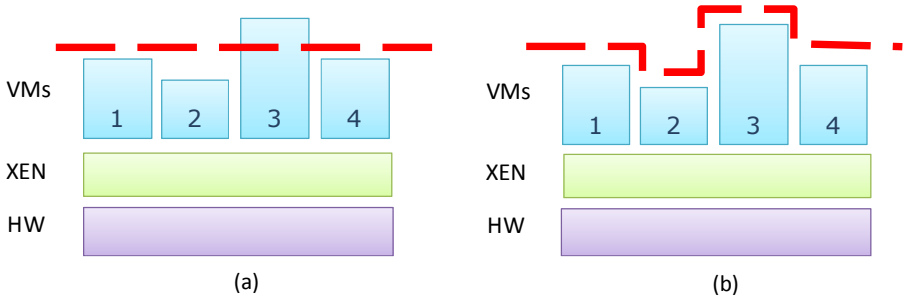

Figura 5 – Distribuição de recursos no Xen

3.1. Caracterização do Problema

Para que a distribuição desses recursos computacionais seja efetuada corretamente, é importante ter conhecimento sobre a demanda de recursos de cada VM e a disponibilidade de recursos para o ambiente como um todo. Para tanto, uma possível estratégia é fazer uso de resultados de testes de desempenho sobre cada VM, os quais podem ser obtidos através da execução de *benchmarks* sobre cada sistema operacional virtualizado. Dada a diversidade de configurações que possam ocorrer nesses ambientes, pode-se ter um grande volume de resultados de *benchmarks*. Examinar, interpretar e aferir esses resultados manualmente para

que, a partir deles, seja possível apontar uma adequada distribuição de recursos, não é uma tarefa trivial de ser realizada. Nesse sentido, é conveniente empregar algum suporte computacional que enderece e auxilie esse processo.

3.2. Caracterização da Contribuição

O objetivo desta pesquisa é propor um processo de apoio à reconfiguração de ambientes virtualizados, baseado em um processo de KDD. O processo de KDD construído é ilustrado na Figura 6. Este processo abrange um total de 12 etapas, conforme segue. Primeiramente são executados *benchmarks* sobre distintas configurações do ambiente virtualizado pelo Xen (a), de onde se obtém dados de configuração e desempenho (b) desse ambiente. Após passarem por um processo de extração, transformação e carga (c), estes dados são armazenados em um DW especificamente projetado para este contexto (d). A partir desse DW, os dados precisam ser preparados (e) para serem utilizados por algoritmos de mineração de dados (f). Estes algoritmos buscam identificar padrões e apontar, através de modelos preditivos (g), características quanto ao desempenho computacional do ambiente virtualizado. Com efeito, esses modelos são enriquecidos (h) para que, ao serem lidos por um subsistema de reconfiguração (i) apresentem, dada uma configuração vigente de uma máquina Xen (j), e seguindo possíveis políticas de SLA (*service level agreement*) (k), qual a melhor maneira de reconfigurar os seus parâmetros (l). Para melhor explicar esse processo de KDD, estas etapas são agrupadas nos seguintes contextos: fonte de dados (etapas a e b); *data warehouse* (etapas c e d); mineração de dados (etapas de e à g); apresentação (etapa h); e efetivação da reconfiguração (etapas de i à l).

Com esse processo, a contribuição desta pesquisa pode ser vista sobre dois aspectos. O primeiro, o *data warehousing* de métricas a partir de *benchmarks* (etapas de a à d). O segundo, a descoberta de padrões para a mineração, onde as métricas armazenadas são convenientemente utilizadas para, a partir delas, poder predizer características que enderecem a reconfiguração de um dado um ambiente (etapas de e à h).

Cabe ressaltar que esta pesquisa está inserida em um projeto de parceria, o qual conta com a soma de esforços entre duas pesquisas complementares que convergem em direção à reconfiguração dinâmica do Xen. Nesse sentido, a efetivação da reconfiguração, que cobre os esforços correspondentes às etapas de (i) à (l) da Figura 6, são objeto da pesquisa proposta por [ROS07]. Por se tratarem de pesquisas complementares, tais etapas são endereçadas para

retratar a relação de dependência que apresentam com o restante do processo. A pesquisa de [ROS07] apresenta o desenvolvimento de um subsistema de reconfiguração que monitora uma dada máquina Xen e verifica se a mesma alerta alguma situação para reconfiguração. Baseado nos resultados dos modelos preditivos gerados, o subsistema infere sobre a máquina Xen vigente alterando, em tempo de execução, seus parâmetros com as instruções recomendadas.

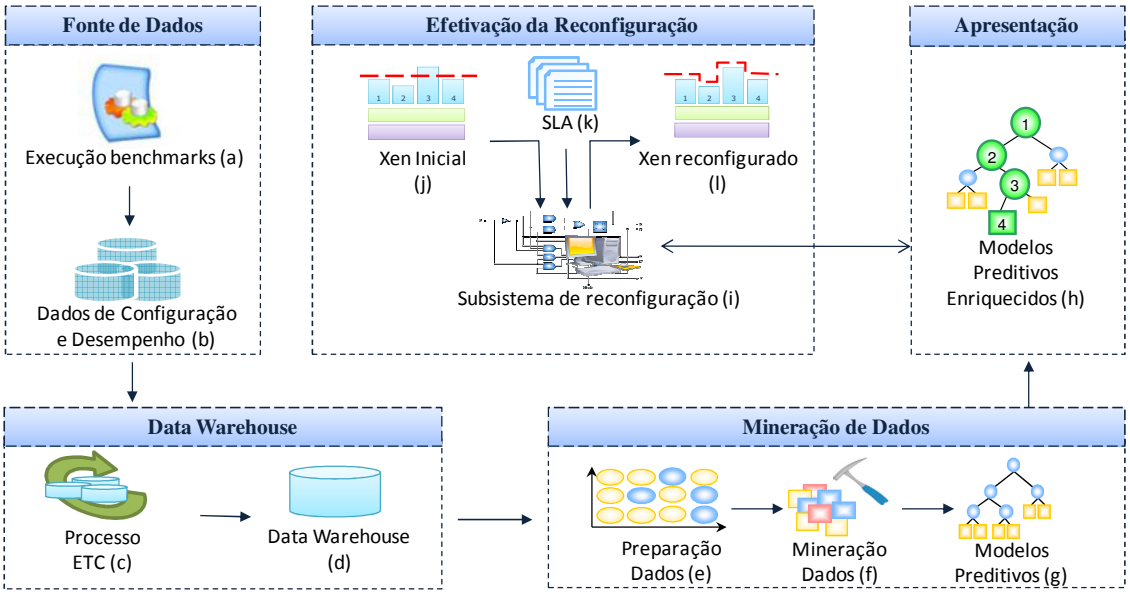

Figura 6 – Processo de KDD construído para a reconfiguração de máquinas Xen

3.3. Considerações do Capítulo

Neste capítulo foi apresentado o cenário desta pesquisa, mostrando como é feita a alocação de recursos para as VMs do Xen, e mostrando qual o cenário ideal de distribuição desses recursos para melhorar seu desempenho. Nesse sentido, o problema consiste em como capturar dados de desempenho e, a partir deles, inferir sobre a correta distribuição dos recursos. Para tanto, como contribuição da pesquisa, foi apresentado um processo completo de KDD para auxiliar a reconfiguração do paravirtualizador Xen. Este processo trata questões de *data warehousing* para métricas de *benchmarks* e de descoberta de padrões para a reconfiguração. Estas etapas são abordadas, respectivamente, nos dois capítulos seguintes, onde são apresentados os métodos utilizados e seus respectivos resultados.

4. DATA WAREHOUSING DE MÉTRICAS DE BENCHMARKS

O processo de *data warehousing* de métricas de *benchmarks* busca permitir que sejam carregadas, no DW desenvolvido, distintas métricas de diferentes execuções de *benchmarks*. Esse *data warehousing* é representado pelos contextos de Fonte de Dados e de *Data Warehouse* (que englobam as etapas de a à d) do processo de KDD proposto (Figura 6), os quais são relatados nesse capítulo. São apresentados os métodos para planejamento de execuções de *benchmark* e para a modelagem do DW. Para cada método são apresentados os testes realizados, com dados desta pesquisa que corroboram a adequação do método proposto.

4.1. Fonte de Dados

As etapas que compõem o processo de KDD construído atuam sobre os dados obtidos das execuções de *benchmarks* e sobre os cenários em que estes foram executados. São esses conjuntos de dados que formam a fonte de dados deste processo de KDD.

As execuções de *benchmarks* buscam simular situações reais, mapeando circunstâncias que possam ocorrer quando o sistema estiver operando com aplicações nas VMs. É preciso que se obtenha um número satisfatório e consistente de resultados de *benchmarks*, a fim de mapear o maior número de situações possíveis. Para tanto, esses são executados em larga escala e sobre diferentes cenários.

4.1.1. Método

As execuções devem ser planejadas buscando englobar um conjunto de configurações que representam o ambiente analisado. No caso da avaliação de ambientes virtualizados, devem ser catalogadas, tipicamente, configurações referentes ao sistema operacional, ao hardware, ao paravirtualizador e ao próprio *benchmark* utilizado. Além disso, devem ser definidas configurações que dizem respeito às próprias execuções, ou seja, que buscam simular aplicações sendo executadas nos sistemas operacionais virtualizados. Assim, respeitando as configurações de ambiente, o planejamento de execução de *benchmarks* deve representar:

- **Limites de Consumo de CPU –** esse limite objetiva simular um ambiente cujas aplicações estejam consumindo um determinado percentual de CPU disponível para o ambiente. Como exemplo, uma VM pode ter aplicações que, ao total, consomem apenas 85% do total de CPU disponível. Assim, deve ser especificado um conjunto limitado de valores que seja capaz de representar aplicações sendo executadas;
- **Total de VMs –** deve ser definido o total de VMs que são carregadas para um dado ambiente;
- **Limites de CAP –** o percentual de CPU definido acima é válido para o ambiente como um todo. Dessa forma, cada VM recebe uma fatia desse total. A essa fatia é atribuído o nome de CAP (topo). Valores de CAP simulam diferentes necessidades de consumo das aplicações que estejam executando em cada VM. É preciso definir um valor mínimo de CAP, bem como intervalos entre esses valores. Posteriormente, é interessante que sejam definidas todas as combinações possíveis de CAP para cada percentual de CPU;
- **Limite de Memória –** além de valores de CAP, para cada VM deve ser alocada uma fatia do total de memória disponível para o ambiente como um todo. Para tanto, é conveniente que também seja adotada alguma estratégia de distribuição; e
- **Número de Configuração –** cada conjunto de configurações deve ser identificado com um número distinto de configuração.

Os *benchmarks* são executados em cada VM de uma dada configuração. Em outras palavras, cada configuração suporta, pelo menos, tantas execuções de *benchmarks* quanto o número de VMs alocadas. Um mesmo *benchmark* pode ser executado diversas vezes sobre uma mesma configuração, assim como podem ser executados *benchmarks* distintos. Cada *benchmark* reporta o os valores encontrados para cada métrica no dado instante de tempo em que foram executados.

4.1.2. Teste

Para atender às necessidades do projeto de parceria que esta pesquisa se enquadra, o planejamento das configurações segue o seguinte critério:

- Para um único ambiente de execução, foram definidos percentuais de CPU que variam entre 70% e 100%, com intervalos de 5 em 5. Adotou-se essa política por considerar-se que, dadas as características do Xen, não haveria aplicações que consumissem menos de 70% do total de CPU disponível;
- Para todos os casos, para atender ao projeto de parceria, são adotadas 4 VMs;
- Para os valores de CAP, buscou-se efetuar todas as possíveis combinações, respeitando um valor mínimo de 10 para cada VM, aumentando esse valor em escalas de 5. A Tabela 2 apresenta o número de combinações definidas para cada limite de CPU; Para exemplificar como essa distribuição foi efetuada, a Tabela 3 mostra as combinações definidas para 70% de CPU; e
- O ambiente definido para os testes conta com 280 MB a serem distribuídos entre as VMs. A distribuição dessa memória obedece a uma estratégia, definida pelo projeto de parceria, que limita em sete combinações distintas, conforme exemplo da Tabela 4. Nessa tabela, para cada VM é ilustrada as diferentes combinações de memória definidas.

Tabela 2 – Número de combinações de CAP por percentual de CPU

Percentual de CPU	70%	75%	80%	85%	90%	95%	100%	Total
Número de Combinações de CAP	9	11	15	18	23	26	34	136

Tabela 3 – Combinações de CAP para 70% de CPU

Combinação	VM1	VM2	VM3	VM4
1	10	10	10	40
2	10	10	15	35
3	10	10	20	30
4	10	15	15	30
5	10	10	25	25
6	10	15	20	25
7	10	20	20	20
8	15	15	15	25
9	15	15	20	20

Tabela 4 – Estratégia de distribuição de memória para 4 VMs

	VM1	VM2	VM3	VM4
Memória (MB)	70	70	70	70
	80	70	70	60
	90	70	70	50
	100	70	70	40
	110	70	60	40
	120	70	50	40
	130	70	40	40

As combinações dos valores de CAP e de memória são alocadas para cada VM conforme exemplo das Tabelas 3 e 4, optando por não realizar a permutação desses valores nas VMs (ver Seção 5.2). Assim, dadas as 136 combinações de CAP e considerando que para cada uma dessas combinações há 7 distribuições de memória, puderam ser planejadas um total de 952 configurações distintas para execuções de *benchmarks*. Como os *benchmarks* são executados sobre cada VM, e cada configuração conta com 4 VMs, esse planejamento permite que obtenha-se um total 3.808 execuções de *benchmarks*. A Tabela 5 exemplifica uma pequena parte desse planejamento, representado por:

- **Ambiente –** são definidas variações de configuração, como memória disponível, escalonador utilizado, hardware, entre outros;
- **CPU –** é utilizados 85% de CPU
- **VMs –** o planejamento conta com 4 VMs paralelas;
- **CAP –** para cada VM é atribuído um valor de CAP, cujas combinações de valores são: (10, 10, 20, 45) e (10, 15, 25, 35);
- **Memória –** o planejamento de distribuição de memória apresentado na Tabela 2 é alocado para cada VM;
- **Configuração –** São definidas 14 configurações (de 239 a 245 e de 281 a 287). Para cada célula de configuração vs. memória há uma execução de *benchmark* (simular ao apresentado na Figura 2, capítulo 2) que retorna métricas e seus respectivos valores para o contexto analisado.

Tabela 5 – Exemplo de planejamento de execuções de benchmarks

Ambiente		Escalonador: Credit / Xen: 3.0.4 / Máquina: Xeon / Kernel: 2.6.16.33 / Memória Disponível: 280MB / Benchmark: Unixbench 3.11								
CPU		85 %				**CPU (%)**		85 %		
VMs		VM 1	VM 2	VM 3	VM 4	**VMs**		VM 1	VM 2	VM 3	VM 4
CAP		10	10	20	45	**CAP**		10	15	25	35
		Memória (MB)						**Memória (MB)**			
Configuração	**239**	70	70	70	70	**Configuração**	**281**	70	70	70	70
	240	80	70	70	60		**282**	80	70	70	60
	241	90	70	70	50		**283**	90	70	70	50
	242	100	70	70	40		**284**	100	70	70	40
	243	110	70	60	40		**285**	110	70	60	40
	244	120	70	50	40		**286**	120	70	50	40
	245	130	70	40	40		**287**	130	70	40	40

É importante ressaltar que o *benchmark* adotado para tais configurações foi o Unixbench. A escolha do mesmo se deu por dois motivos: (1) é opção do projeto de parceria trabalhar apenas com métricas de CPU e de memória (as métricas do Unixbench atendem a essa características); e (2) as execuções de *benchmarks* para cada configuração demandam muito tempo (não haveria tempo hábil para executar mais de um *benchmark*, como LmBench e Iozone, em cada configuração). Além disso, apesar do planejamento, puderam ser executados *benchmarks* apenas para 70%, 85%, 90% e 100% de CPU e, para os dois últimos, as execuções se deram para um total de 21 e 29 combinações de CAP, respectivamente. Assim, têm-se um total de 77 combinações de CAP que geraram 539 configurações, totalizando em 2.156 execuções de *benchmarks*.

4.2. Data Warehouse

Para que os dados de execução de *benchmarks* e de suas respectivas configurações sejam adequadamente armazenados, é construído um DW. Assim, a modelagem do DW é uma importante etapa para compor o cenário desse processo de KDD.

4.2.1. Método

O modelo analítico construído é baseado no esquema floco de neve, focalizado em um cenário de captura de métricas de *benchmarks*. Este modelo, ilustrado na Figura 7, possui dimensões que denotam:

(a) **Ambiente –** essas dimensões são modeladas hierarquicamente para atender à diversidade de configurações que possam ocorrer;

(b) **Configuração –** tais dimensões denotam o cenário em que cada execução é efetuada, de modo a indicar qual VM ou conjunto de VMs que está sendo utilizado; e

(c) **Métricas –** dimensões que representam os resultados obtidos das execuções de *benchmarks*.

Entendeu-se ser o esquema floco de neve o mais conveniente, pois o perfil típico do usuário desse modelo domina a estruturação proposta, principalmente, para configuração do ambiente. Para os experimentos realizados, o modelo apresentou-se suficientemente

abrangente, sendo capaz de armazenar uma série de execuções para diferentes configurações e *benchmarks*.

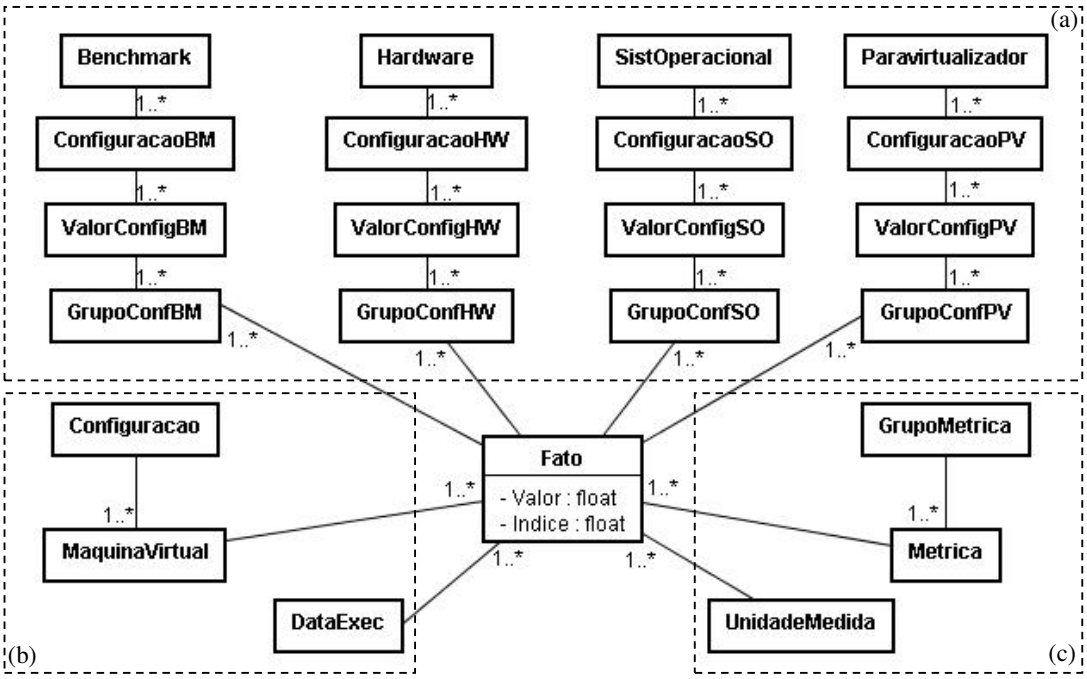

Figura 7 – Modelo analítico para execuções de *benchmarks*

Cada conjunto das dimensões de configuração de ambiente (Figura 7a) é organizado em quatro níveis, onde os três primeiros especificam a configuração de cada característica. Esta solução permite acomodar quaisquer detalhamentos que as propriedades necessitem para melhor caracterizarem os fatos. Para que seja possível associar um fato a uma combinação de configurações, o quarto nível é uma tabela ponte, na terminologia de [KIM98], que agrupa diferentes conjuntos de configurações. Assim, as dimensões de ambiente são organizadas como segue:

- **Propriedade –** dimensões *Benchmark*, Hardware, SistemaOperacional e Paravirtualizador. Os atributos que compõem essas dimensões são: código da propriedade e descrição da propriedade;
- **Configuração da Propriedade –** dimensões ConfiguracaoBM, ConfiguracaoHW, ConfiguracaoSO e ConfiguracaoPV. Os atributos que compõem essas dimensões são: código da configuração e descrição da configuração, além da referência da propriedade a que pertence;
- **Valores de Configuração de cada Propriedade –** dimensões ValorConfigBM, ValorConfigHW, ValorConfigSO e ValorConfigPV. Os

atributos que compõem essas dimensões são: código do valor de configuração e valor da configuração, além da referência da configuração da propriedade a que pertence; e

- **Grupo –** dimensões GrupoConfBM, GrupoConfHW, GrupoConfSO e GrupoConfPV. É composta pelo código do grupo e referências para valores de configuração de cada propriedade e tabela fato.

As dimensões de configuração (Figura 7b) comportam características referentes ao planejamento efetuado para cada configuração (tais configurações são semelhantes ao planejamento exemplificado na Tabela 5), onde são estabelecidas as três seguintes dimensões:

- **Configuracao –** essa dimensão busca representar as características de cada configuração. Os atributos que compõem essa dimensão são: código da configuração, tipo da configuração, percentual de CPU e total de VM;
- **MaquinaVirtual –** essa dimensão busca representar as características de cada máquina virtual em cada execução, além da referência à configuração. Os atributos que compõe essa dimensão são: número da VM (indicando qual VM está sendo analisada), memória, CAP e média (esse atributo indica o valor referente à *performance* total de cada VM, conforme exemplo da Figura 2 no capítulo 2); e
- **DataExec –** essa dimensão identifica a estampa de tempo (*timestamp*) em que as execuções de *benchmarks* foram efetuadas para cada configuração. Os atributos que compõem essa dimensão são: código e estampa de tempo para uma dada execução.

As dimensões de métricas (Figura 7c) armazenam características de cada métrica, reportada por cada *benchmark*. Tais dimensões são:

- **GrupoMetrica** – Alguns *benchmarks*, como o LmBench, utilizam critérios próprios de agrupamento de métricas, reportando o mesmo nome de métrica em diferentes grupos. Entendeu-se ser conveniente reproduzir esses agrupamentos no modelo para garantir a abrangência do mesmo. Os atributos que compõem essa dimensão são: código do grupo de métricas e descrição do grupo;

- **Metrica –** Para cada registro do grupo de métricas, as mesmas são agrupadas em diferentes contextos, onde essa dimensão contém todas as características referentes às métricas reportadas por cada *benchmark*. Os atributos que compõem essa dimensão são: código da métrica, nome da métrica e tipo da métrica (indicando o aspecto medido como, por exemplo, CPU e memória); e
- **UnidadeMedida** – nessa dimensão são identificadas as unidades de medidas utilizadas para cada métrica, em cada *benchmark*. *Benchmarks* distintos podem possuir um mesmo nome de métrica ou métricas equivalentes, porém com unidades de medida distintas. Os atributos que compõem essa dimensão são: código da unidade de medida, unidade de medida e descrição da unidade de medida.

A tabela fato desse modelo representa os valores medidos para cada métrica em seu respectivo contexto. Este é composto por um identificador, pelos atributos-chave das dimensões, caracterizando o cenário em cada métrica foi capturada, e por dois atributos numéricos que representam o valor medido para cada métrica e seu índice correspondente (estes valores são calculados e fornecidos pelo *benchmark*). Com a estruturação proposta, o DW é capaz de acomodar execuções de diferentes *benchmarks* em diferentes contextos de configurações.

O modelo analítico desenvolvido busca englobar todo o cenário em que as execuções de *benchmarks* são efetuadas. Entretanto, caso haja alguma evolução no processo de execução, outras situações podem ser exploradas. Esse modelo pode evoluir de acordo com os cenários de execuções em que o processo se encontra. Sendo assim, caso haja novas abordagens que não estejam previstas nos contextos de configuração de ambiente, configurações e métricas, o modelo apresentado é capaz de ser estendido ou modificado, sem apresentar ônus para as características já presentes, mantendo os princípios usados na sua concepção.

No que se refere aos resultados reportados pelos *benchmarks*, não há uma uniformização na apresentação dos mesmos, nem um formato padrão para intercâmbio de resultados. Além disso, para que o DW seja alimentado, é necessário complementar os resultados das execuções de *benchmark* com os dados de configuração.

Para garantir a consistência dos dados do DW, é feita uma análise cuidadosa nos valores resultantes das execuções de *benchmarks* e nos parâmetros contidos no planejamento

de execuções, de forma com que os dados dessas duas fontes sejam adequadamente relacionados. Esse processo pode se beneficiar bastante caso tenha um nível de automação maior como, por exemplo, o proposto em [BEC06] e em [SIL07].

4.2.2. Teste

Para as dimensões de ambiente, a conveniência da estruturação em 4 camadas pode ser observada com o exemplo da Tabela 6, a qual reporta exemplos de conteúdo referentes ao Paravirtualizador. O cabeçalho desta tabela apresenta os nomes das dimensões (Paravirtualizador, ConfiguracaoPV, ValorConfigPV e GrupoPV). Para cada dimensão são apontados os seus atributos e exemplos de respectivos registros. Nessa tabela é possível observar que, para um dado paravirtualizador Xen, existem quatro aspectos de configuração, sendo que cada um desses possui uma ou mais variações de valores: há duas diferentes versões (3.0.4 e 3.0.2); três distintos valores para memória utilizada (476 MB, 470 MB e 440 MB); dois possíveis escalonadores (Credit e Sedf); e um valor para a memória do Domínio 0 (196 MB). Essas características ainda podem ser combinadas entre si em diferentes situações. Nesse sentido, a finalidade da utilização de grupos mostra-se mais evidente pois, conforme o exemplo, as configurações podem estar agrupadas em diferentes perspectivas, onde cada uma pode fazer parte de um ou mais grupos, contendo as seguintes configurações:

- **Grupo 01 –** Versão: 3.0.4; Memória: 476 MB; Escalonador: Credit; Domínio0: 196 MB;
- **Grupo 02 –** Versão 3.0.4; Memória: 470 MB; Escalonador: Sedf; Domínio0: 196 MB;
- **Grupo 03 –** Versão 3.0.2; Memória: 440 MB; Escalonador: Sedf; Domínio0: 196 MB;
- **Grupo 04 –** Versão: 3.0.2; Memória: 476 MB; Escalonador: Credit; Domínio0: 196 MB;

Exemplos de conteúdo das dimensões de ambiente relacionadas à Hardware, *Benchmark* e Sistema operacional são ilustrados nas Tabelas 7, 8 e 9, respectivamente. No cabeçalho dessas tabelas estão dispostos os nomes das dimensões, seguido de seus atributos e respectivas instâncias. Na Tabela 7 observa-se que, para uma máquina Xeon, as configurações disponíveis são memória (512 MB) e processador (2.4 GHz). Ambas as configurações são

identificadas pelo grupo 01. A Tabela 8 mostra que, para o *benchmark* Unixbench pode haver duas versões (3.11 e 3.04), as quais pertencem, respectivamente, aos códigos de valores de grupo 01 e 02. A Tabela 9 mostra que, para o sistema operacional Linux, é definida uma configuração de *Kernel* (2.6.16.33), a qual é identificada pelo grupo 01.

Tabela 6 – Exemplo de conteúdos referentes ao paravirtualizador

Paravirtualizador		**ConfiguracaoPV**			**ValorConfigPV**				**GrupoConfPV**			
Cód PV	Paravirtualizador	Cód Conf	Cód PV	Configuração	Cód Valor	Cód Conf	Cód PV	Valor	Grupo PV	Cód Valor	Cód Conf	Cód PV
01	Xen	01	01	Versão	01	01	01	3.0.4	01	01	01	01
		02	01	Memória	01	02	01	476 MB	01	01	02	01
		03	01	Escalonador	01	03	01	Credit	01	01	03	01
		04	01	Dominio0	01	04	01	196 MB	01	01	04	01
					02	01	01	3.0.2	02	01	01	01
					02	02	01	470 MB	02	02	02	01
					02	03	01	Sedf	02	02	03	01
					03	02	01	440 MB	02	01	04	01
									03	02	01	01
									03	03	02	01
									03	02	03	01
									03	01	04	01
									04	02	01	01
									04	01	02	01
									04	01	03	01
									04	01	04	01

Tabela 7 – Exemplo de registros nas dimensões de ambiente de hardware

Hardware		**ConfiguracaoHW**			**ValorConfigHW**				**GrupoConfHW**			
Cód HW	Hardware	Cód Conf	Cód HW	Configuração	Cód Valor	Cód Conf	Cód HW	Valor	Grupo HW	Cód Valor	Cód Conf	Cód HW
01	Xeon	01	01	Memória	01	01	01	512 MB	01	01	01	01
		02	01	Processador	01	02	01	2.4 GHz	01	01	02	01

Tabela 8 – Exemplo de registros nas dimensões de ambiente de benchmark

Benchmark		**ConfiguracaoBM**			**ValorConfigBM**				**GrupoConfBM**			
Cód BM	Benchmark	Cód Conf	Cód BM	Configuração	Cód Valor	Cód Conf	Cód BM	Valor	Grupo BM	Cód Valor	Cód Conf	Cód BM
01	Unixbench	01	01	Versão	01	01	01	3.11	01	01	01	01
					02	01	01	3.04	02	01	01	01

Tabela 9 – Exemplo de registros nas dimensões de ambiente de sistema operacional

Sistema Operacional		**ConfiguracaoSO**			**ValorConfigSO**				**GrupoConfSO**			
Cód SO	Sistema Operacional	Cód Conf	Cód SO	Configuração	Cód Valor	Cód Conf	Cód SO	Valor	Cód Grupo	Cód Valor	Cód Conf	Cód SO
01	Linux	01	01	Kernel	01	01	01	2.6.16.33	01	01	01	01

Exemplos para registros das dimensões de configuração estão ilustrados nas Tabelas 10 e 11. A Tabela 10 mostra exemplos de registros para a dimensão DataExec. Já a Tabela 11 exemplifica um conjunto de dados para as dimensões Configuracao e MaquinaVirtual, de acordo com o exemplo de planejamento da Tabela 5, para as configurações 239, 241, 243, 245, 281, 283, 285 e 287. O cabeçalho da Tabela 11 identifica essas dimensões, seguido de seus atributos e respectivas instâncias. Para cada registro da dimensão Configuracao há quatro registros na dimensão MaquinaVirtual.

Tabela 10 – Exemplo de registros para a dimensão DataExec

DataExec	
Código DataExec	**Estampa de Tempo**
01	2007/05/16
02	2007/06/21
03	2007/09/19

Tabela 11 – Exemplos de registros para as dimensões Configuracao e MaquinaVirtual

Configuracao				MaquinaVirtual				
Código Config	Tipo Configuração	% CPU	Total VM	Código Config	Número VM	CAP	Me-mória	Média
239	CPU: 85% CAP: 10 10 20 45	85	4	239	1	10	70	14
241	CPU: 85% CAP: 10 10 20 45	85	4	239	2	10	70	14,1
243	CPU: 85% CAP: 10 10 20 45	85	4	239	3	20	70	24,3
245	CPU: 85% CAP: 10 10 20 45	85	4	239	4	45	70	48,9
281	CPU: 85% CAP: 10 15 25 35	85	4	241	1	10	90	14,3
283	CPU: 85% CAP: 10 15 25 35	85	4	241	2	10	70	14
285	CPU: 85% CAP: 10 15 25 35	85	4	241	3	20	70	25,4
287	CPU: 85% CAP: 10 15 25 35	85	4	241	4	45	50	48,7
				243	1	10	110	14,3
				243	2	10	70	14,3
				243	3	20	60	24,8
				243	4	45	40	49,1
				245	1	10	130	15
				245	2	10	70	14,7
				245	3	20	40	23,9
				245	4	45	40	48,9
				281	01	10	70	14,3
				281	02	15	70	21,0
				281	03	25	70	30,2
				281	04	35	70	41,1
				283	01	10	90	14,3
				283	02	15	70	21,4
				283	03	25	70	31,1
				283	04	35	50	39,1
				285	01	10	110	14,7
				285	02	15	70	22,1
				285	03	25	60	30,4
				285	04	35	40	39,3
				287	01	10	130	14,8
				287	02	15	70	21,5
				287	03	25	40	30,6
				287	04	35	40	39,7

Para as dimensões de métricas, os exemplos de registros são apresentados nas Tabelas 12 e 13. A Tabela 12 ilustra exemplos de registros para a dimensão UnidadeMedida, os quais reportam as unidades de medida utilizadas pelo Unixbench. O cabeçalho dessa tabela indica o nome da dimensão e os atributos que compõe a mesma. Já a Tabela 13 exemplifica um conjunto de registros para as dimensões GrupoMetrica e Metricas. O cabeçalho dessa tabela indica o nome da dimensão, seguido de seus atributos e respectivas instâncias. No caso do exemplo, são ilustradas apenas as 6 métricas do Unixbench que foram reportadas no resumo das informações deste *benchmark* (Figura 2c). Como esse *benchmark* não faz uso de agrupamentos, o nome do grupo de métricas foi definido com o nome do próprio *benchmark*.

Tabela 12 – Exemplos de registros para a dimensão UnidadeMedida

UnidadeMedida		
Código Un Med	**Unidade Medida**	**Descrição**
01	lps	Loops Per Second
02	lpm	Loops Per Minute
03	Kbps	Kbytes Per Second

Tabela 13 – Exemplos de registro para as dimensões de métricas

GrupoMetricas		Metricas			
Código Grupo	**Nome Grupo**	**Código Grupo**	**Código Métrica**	**Nome Métrica**	**Tipo Métrica**
01	Unixbench	01	02	Arithmetic Test (type = double)	CPU
		01	11	Dhrystone 2 without register variables	CPU
		01	12	Execl Throughput Test	Memória
		01	14	File Copy (30 seconds)	Memória
		01	20	Pipe-based Context Switching Test	CPU
		01	26	Shell scripts (8 concurrent)	CPU

Com base nos registros das dimensões, a Tabela 14 mostra exemplo de registros para a Tabela Fato. Para o exemplo, foram inseridos 48 registros correspondentes às configurações de número 239 e 281. No exemplo, cada configuração conta com 24 registros na tabela Fato, uma vez que estão sendo relacionadas, para cada uma das 4 VMs, as 6 métricas ilustradas na Tabela 13.

Dadas as 2.156 execuções do *benchmark* Unixbench, considerando que cada resultado desse *benchmark* produziu um total de 27 métricas e seus respectivos valores, a Tabela Fato com os dados desta pesquisa conta com um total de 58.212 registros.

Tabela 14 – Exemplo de registros para a tabela fato

Tabela Fato											
Código Fato	Grupo BM	Grupo HW	Grupo SO	Grupo PV	Código Config	Numero VM	Código DataExec	Código Metrica	Código Un Med	Valor	Índice
01	01	02	01	01	239	01	01	02	01	54760.0	21.5
02	01	02	01	01	239	01	01	11	01	349888.7	15.6
03	01	02	01	01	239	01	01	12	01	125.9	7.6
04	01	02	01	01	239	01	01	14	03	5238.0	29.3
05	01	02	01	01	239	01	01	20	01	6267.7	4.8
06	01	02	01	01	239	01	01	26	02	21.0	5.2
07	01	02	01	01	239	02	01	02	01	55092.6	21.7
08	01	02	01	01	239	02	01	11	01	352507.4	15.8
09	01	02	01	01	239	02	01	12	01	127.4	7.7
10	01	02	01	01	239	02	01	14	03	5318.0	29.7
11	01	02	01	01	239	02	01	20	01	6271.3	4.8
12	01	02	01	01	239	02	01	26	02	21.0	5.2
13	01	02	01	01	239	03	01	02	01	110883.0	43.6
14	01	02	01	01	239	03	01	11	01	704067.0	31.5
15	01	02	01	01	239	03	01	12	01	252.4	15.3
16	01	02	01	01	239	03	01	14	03	6362.0	35.5
17	01	02	01	01	239	03	01	20	01	12658.6	9.6
18	01	02	01	01	239	03	01	26	02	42.0	10.5
19	01	02	01	01	239	04	01	02	01	256325.9	100.8
20	01	02	01	01	239	04	01	11	01	1533585.7	68.6
21	01	02	01	01	239	04	01	12	01	593.2	36.0
22	01	02	01	01	239	04	01	14	03	7640.0	42.7
23	01	02	01	01	239	04	01	20	01	29236.6	22.2
24	01	02	01	01	239	04	01	26	02	92.3	23.1
25	01	02	01	01	281	01	01	02	01	54802.9	21.6
26	01	02	01	01	281	01	01	11	01	351646.0	15.7
27	01	02	01	01	281	01	01	12	01	126.0	7.6
28	01	02	01	01	281	01	01	14	03	5533.0	30.9
29	01	02	01	01	281	01	01	20	01	6267.4	4.8
30	01	02	01	01	281	01	01	26	02	21.0	5.2
31	01	02	01	01	281	02	01	02	01	92619.1	36.4
32	01	02	01	01	281	02	01	11	01	593690.6	26.5
33	01	02	01	01	281	02	01	12	01	212.7	12.9
34	01	02	01	01	281	02	01	14	03	5978.0	33.4
35	01	02	01	01	281	02	01	20	01	10592.2	8.0
36	01	02	01	01	281	02	01	26	02	35.0	8.8
37	01	02	01	01	281	03	01	02	01	147895.7	58.2
38	01	02	01	01	281	03	01	11	01	947739.9	42.4
39	01	02	01	01	281	03	01	12	01	338.8	20.5
40	01	02	01	01	281	03	01	14	03	5927.0	33.1
41	01	02	01	01	281	03	01	20	01	16767.7	12.7
42	01	02	01	01	281	03	01	26	02	56.0	14.0
43	01	02	01	01	281	04	01	02	01	200486.9	78.9
44	01	02	01	01	281	04	01	11	01	1279993.8	57.2
45	01	02	01	01	281	04	01	12	01	458.4	27.8
46	01	02	01	01	281	04	01	14	03	8331.0	46.5
47	01	02	01	01	281	04	01	20	01	22869.0	17.3
48	01	02	01	01	281	04	01	26	02	75.3	18.8

4.3. Considerações do Capítulo

Este capítulo apresentou o *data warehousing* do processo de KDD construído para reconfiguração do Xen. Foi mostrado como é efetuado um planejamento de execuções para obter resultados de diferentes cenários de configuração de ambiente. Foi construído um DW

focalizado em captura de métricas de *benchmarks* para acomodar os diferentes valores de métricas reportados por cada *benchmark*, em cada cenário de execução. Esse modelo, para os testes realizados, mostrou-se abrangente, sendo capaz de acomodar todos os dados planejados e necessários para esta pesquisa. Fazendo uso desses dados organizados, o capítulo seguinte apresenta como a mineração de dados pode ser útil em sugerir parâmetros de reconfiguração para o Xen.

5. DESCOBERTA DE PADRÕES PARA A RECONFIGURAÇÃO

Os contextos de mineração de dados e de apresentação, que englobam as etapas de (e) à (h) do processo de KDD proposto (Figura 6), caracterizam a descoberta de padrões para a mineração. Essa última tira proveito da organização dos dados de execução de *benchmarks* no DW para, a partir deles, extrair padrões que auxiliem na reconfiguração de ambientes virtualizados (contexto de efetivação da reconfiguração, etapas de (i) à (l) da Figura 6). Neste capítulo são relatadas as etapas de mineração de dados e de apresentação, enfatizando a preparação de dados para a geração de modelos preditivos, e como esses modelos gerados são enriquecidos para que possam ser utilizados pelo subsistema de reconfiguração. Para cada etapa são apresentados os métodos aplicados e respectivos testes realizados.

5.1. Mineração de Dados

A mineração de dados, nesta pesquisa, é aplicada para identificar se é possível melhorar a *performance* de uma dada máquina Xen, e como isso deve ser feito, indicando uma reconfiguração adequada para seus parâmetros. São utilizadas tarefas preditivas, focalizadas em algoritmos de classificação, para indicar, dada uma configuração de uma máquina Xen vigente, quais novos conjuntos de configuração podem fornecer melhora de desempenho. Como esta pesquisa não tem por objetivo desenvolver um novo algoritmo de mineração de dados para gerar os resultados esperados, para testar essa abordagem é utilizado, como sistema de apoio, o algoritmo J48 do ambiente Weka. Assim, os esforços empregados concentram-se na preparação dos dados para a mineração, a fim de que os mesmos contribuam para uma adequada produção de modelos preditivos que auxiliem na reconfiguração do Xen.

5.1.1. Método

Por estar sendo utilizada uma tarefa preditiva, optou-se em realizar a preparação dos dados para a mineração em duas etapas. Na primeira, os dados são preparados para definir o atributo preditivo. Na segunda, os atributos explanatórios são preparados e organizados em

um arquivo do tipo arff para serem lidos pelo algoritmo de mineração. Em ambas as etapas, faz-se uso conveniente das técnicas de preparação discutidas na Seção 2.2.4.

5.1.1.1. Definição do Atributo Preditivo

Para definir o atributo preditivo, considera-se necessário efetuar um mapeamento entre todas as configurações, de modo a identificar se a alteração de uma dada configuração para outra traz vantagens, ou não, na reconfiguração. Esse mapeamento é feito considerando o desempenho global de cada configuração, como segue:

- Primeiramente são coletados todos os valores do atributo média da dimensão MaquinaVirtual do DW (ver Seção 4.2.1);
- Para obter o desempenho global, é aplicada a técnica de transformação de atributos, efetuando o somatório do atributo média de todas as VMs que fazem parte de uma dada configuração;
- A partir desse somatório, cada configuração é comparada com as demais, duas a duas, onde se tem uma configuração inicial e uma configuração alvo (ex: tendo a configuração 1 como inicial, tem as configurações 2, 3, ..., n como alvo); e
- Aplicando a técnica de criação de atributos, a comparação entre as configurações é feita com algum critério definido como benefício. No caso, como ponto de partida, pode-se usar a diferença entre o somatório de cada configuração alvo e o somatório da configuração inicial.

De posse desses resultados, é elaborada uma matriz quadrada, para todas as configurações estabelecidas, cujo objetivo é mapear o efeito da mudança de configurações. As células dessa matriz contêm valores 0 ou 1. Nesse caso é aplicada a técnica de transformação de atributos, verificando o resultado da diferença entre a configuração alvo e a configuração inicial. Caso a diferença seja maior do que zero, significa que houve melhora de desempenho, e a célula correspondente da matriz é alimentada com 1; caso a diferença seja menor ou igual a zero, significa que a configuração inicial apresenta um desempenho melhor ou equivalente à configuração alvo, e a célula correspondente da matriz é alimentada com 0. O valor de cada célula da matriz vai dizer se mudança da configuração inicial (eixo x), para a configuração alvo (eixo y) é benéfica, e esse será o atributo preditivo. Como efeito, essa matriz deve ser

capaz de produzir um grafo dirigido, necessariamente acíclico, com as mudanças de configurações que são benéficas.

5.1.1.2. Preparação dos Atributos Explanatórios

Como a definição do atributo preditivo é efetuada considerando-se uma configuração inicial comparada com as configurações alvo, é apropriado que a preparação dos demais atributos, os quais irão compor o arquivo para a mineração, siga o mesmo princípio. Por questões de simplicidade e portabilidade, os dados devem ser dispostos em um arquivo do tipo arff. Esse arquivo deve conter, em cada registro, dados que representam a configuração inicial, dados que representam a configuração alvo, e o valor do atributo preditivo correspondente. A preparação desses atributos segue os seguintes critérios:

- Os atributos que compõem o arquivo arff devem ser aqueles que correspondem aos parâmetros de configuração que tem-se interesse de alterar: caso não seja objetivo alterar configurações de hardware, atributos desse tipo não devem ser utilizados. Nesse caso é aplicada a técnica de seleção de atributos;
- O arquivo ainda pode conter apenas os atributos selecionados que atendam à uma determinada categoria. Como exemplo, pode-se querer apenas os atributos que correspondam às execuções de 85% de CPU. Para selecionar apenas essas instâncias, é aplicada a redução de dimensionalidade;
- É conveniente verificar se os valores dos atributos selecionados, quando numéricos, merecem ser tratados como valores ordenáveis. Caso não seja conveniente que os mesmos apresentem alguma relação de ordem, aplica-se a técnica de discretização para que esses se tornem categóricos; e
- Alguns atributos podem ser utilizados individualmente ou tratados como a combinação de seus valores para a configuração (ex: CAP e memória). Se não for conveniente representá-los individualmente, aplica-se a técnica de criação de atributos, a fim de definir valores que representem cada combinação.

Seguindo esse procedimento, o algoritmo de mineração aplicado sobre esse arquivo pode, então, ser capaz de produzir modelos preditivos que indiquem quais parâmetros de configuração, se alterados, podem representar melhora de desempenho para uma dada configuração.

5.1.1.3. Interpretação dos Modelos Preditivos Produzidos

A saída do classificador é uma árvore de um ou mais níveis onde podem aparecer valores para um ou mais atributos explanatórios e, no último nível, constam 1 ou 0, identificando se a reconfiguração é benéfica ou não. Neste tipo de saída, os atributos explanatórios que não aparecem na mesma, são considerados irrelevantes pelo classificador para predizer o atributo preditivo.

A qualidade dos modelos pode ser medida não apenas pela acurácia dos mesmos, mas analisando os valores apresentados na matriz de confusão. Embora os valores de VP e VN representem as situações que foram classificadas corretamente, os valores de FN e FP também devem ser analisados. Os falsos negativos produzidos pelo modelo não prejudicam a qualidade da máquina Xen na reconfiguração, pois o subsistema apenas não fará uso dessas reconfigurações. Por outro lado, os falsos positivos, para o problema endereçado, podem ser considerado como o erro de classificação mais crítico. Esses erros podem induzir o subsistema em reconfigurar uma dada máquina Xen para alguma configuração desvantajosa. Nesse sentido, um aprimoramento na definição do benefício em uma alteração de configuração pode endereçar essa questão.

5.1.2. Teste

Para o teste da etapa de mineração, são utilizados os dados do teste para o DW, apresentados no capítulo anterior. Para tanto, é preparado o atributo preditivo, gerado o arquivo arff, executado o algoritmo e analisada a credibilidade da solução.

5.1.2.1. Atributo Preditivo

Para a preparação do atributo preditivo, são coletados os valores do atributo média de cada VM das 539 configurações definidas e, para cada configuração, é efetuado o somatório das médias das VMs. Para exemplificar, a Tabela 15 ilustra resultados desse processo para as configurações 239, 241, 243, 245, 281, 283, 285 e 287. Todas essas configurações são comparadas entre si. Essa tabela está dividida em configuração inicial, configuração alvo e resultado. Os valores de somatório correspondem à soma dos valores do atributo média de cada VM das configurações citadas. Para fins de simplificação, os valores para cada VM não

estão reportados nessa tabela, mas podem ser conferidos na Tabela 11. O resultado apresenta a diferença entre o somatório da configuração alvo e o somatório da configuração inicial. Nas transições cujo valor da diferença é maior que zero, o atributo preditivo correspondente é 1 e, caso contrário é 0.

Com os resultados dos atributos preditivos mapeados para cada configuração definida, é produzida uma matriz quadrada de tamanho 539. Para exemplificar o efeito das transições dessa matriz, a Figura 8 ilustra um grafo acíclico dirigido para as configurações apresentadas na Tabela 15. As arestas do grafo indicam as transições que apresentam melhora de desempenho. Nota-se que o grafo tem um único ponto de partida: a configuração 239, o que significa que, no exemplo, o desempenho desta configuração sempre pode ser melhorado. Por outro lado, os nodos 281 e 287 são os pontos finais do grafo. Essas duas configurações são vistas como ideais nesse contexto, não podendo ser melhoradas. A transitividade entre as transições não estão representadas no grafo.

Tabela 15 – Exemplo de preparação do atributo preditivo

Configuração Inicial		**Configuração Alvo**		**Resultado**	**Configuração Inicial**		**Configuração Alvo**		**Resultado**
Código Config	Soma-tório	Código Config	Soma-tório	Diferença	Código Config	Soma-tório	Código Config	Soma-tório	Diferença
239	101,3	241	102,4	1,1	281	106,6	239	101,3	-5,3
		243	102,5	1,2			241	102,4	-4,2
		245	102,5	1,2			243	102,5	-4,1
		281	106,6	5,3			245	102,5	-4,1
		283	105,9	4,6			283	105,9	-0,7
		285	105,7	4,4			285	105,7	-0,9
		287	106,6	5,3			287	106,6	0
241	102,4	239	101,3	-1,1	283	105,9	239	101,3	-4,6
		243	102,5	0,1			241	102,4	-3,5
		245	102,5	0,1			243	102,5	-3,4
		281	106,6	4,2			245	102,5	-3,4
		283	105,9	3,5			281	106,6	0,7
		285	105,7	3,3			285	105,7	0,2
		287	106,6	4,2			287	106,6	0,7
243	102,5	239	101,3	-1,2	285	105,7	239	101,3	-4,4
		241	102,4	-0,1			241	102,4	-3,3
		245	102,5	0			243	102,5	-3,2
		281	106,6	4,1			245	102,5	-3,2
		283	105,9	3,4			281	106,6	-0,9
		285	105,7	3,2			283	105,9	-0,2
		287	106,6	4,1			287	106,6	-0,9
245	102,5	239	101,3	-1,2	287	106,6	239	101,3	-5,3
		241	102,4	-0,1			241	102,4	-4,2
		243	102,5	0			243	102,5	-4,1
		281	106,6	4,1			245	102,5	-4,1
		283	105,9	3,4			281	106,6	0
		285	105,7	3,2			283	105,9	0,7
		287	106,6	4,2			285	105,7	0,9

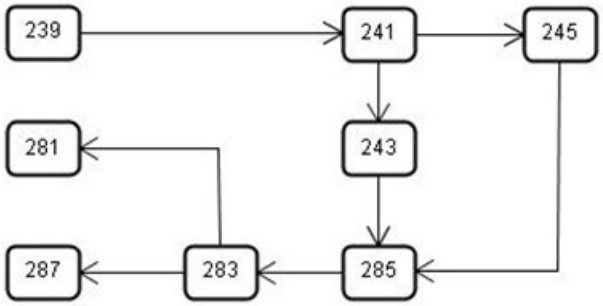

Figura 8 – Exemplo de transição entre configurações

5.1.2.2. Geração do Arquivo Arff

O projeto de parceria desta pesquisa demanda que a reconfiguração do Xen seja feita em tempo de execução. Nesse sentido, o arquivo arff é preparado apenas com os atributos dos parâmetros que podem ser alterados dinamicamente, podendo-se escolher os seguintes: percentual de CPU, CAP e memória. Entretanto, também em função das necessidades do projeto, as instruções para a reconfiguração são realizadas respeitando o percentual de CPU de origem. Assim, são preparados arquivos arff distintos para cada percentual de CPU definido, onde os atributos selecionados são CAP e memória, para as configurações iniciais e alvo.

Com uma série de experimentos efetuados, detectaram-se algumas perturbações. Essas mostraram que os atributos CAP e memória, por serem numéricos, estavam sendo tratados, pelo algoritmo de mineração, como valores ordenáveis. Contudo, constatou-se que os mesmos não apresentam relação de ordem, necessariamente, na maioria dos casos. Assim, optou-se por discretizá-los, tornando todos categóricos.

Além disso, os valores de CAP são armazenados individualmente para cada VM. Como cada combinação de valores de CAP é única, é interessante que o CAP seja trabalhado como a combinação de valores para cada configuração, e não tratados individualmente. Assim, aplicando as técnicas de criação de atributos e discretização, foi definido um atributo categórico para cada combinação de CAP. O conteúdo desse atributo é correspondente à primeira configuração pertencente a uma dada combinação de CAP (ex: conforme apresentado na Tabela 5, a combinação de CAP (10, 15, 25, 35) atende às configurações de 281 a 287. Logo, o valor categórico para o atributo que representará essa combinação de CAP é 281). De maneira análoga, os valores de memória para as VM são tratados como um

conjunto, e não individualmente. Aplicando as técnicas de seleção de atributos e discretização, para cada combinação de memória é assumido o valor da memória da primeira VM (ex: no caso da combinação de memória (110, 70, 60, 40), o valor categórico para o atributo que representará tal combinação de memória é 110).

A Tabela 16 apresenta os percentuais de CPU que produziram execuções de *benchmarks* (ver seção 4.1.2), o nome do arquivo arff correspondente, o número de combinações de CAP e o número de instâncias em cada arquivo, a qual corresponde a a $((C * 7)^2)$, onde (C) é número de combinações de CAP para um dado percentual de CPU.

Tabela 16 – Número de instâncias dos arquivos arff

Percentual de CPU	Nome do Arquivo Arff	Comibinações de CAP (C)	Número de Instâncias $(C * 7)^2$
70%	Arff_70	9	3.969
85%	Arff_85	18	15.876
90%	Arff_90	21	21.609
100%	Arff_100	29	41.209

A Figura 9 mostra uma versão simplificada da estruturação do arquivo Arff_85 preparado, com alguns dados selecionados que correspondem às configurações subseqüentes:

- **Configuração 239 –** CAP 239 e Memória 70;
- **Configuração 241 –** CAP 239 e Memória 90;
- **Configuração 243 –** CAP 239 e Memória 110;
- **Configuração 245 –** CAP 239 e Memória 130;
- **Configuração 281 –** CAP 281 e Memória 70;
- **Configuração 283 –** CAP 281 e Memória 90;
- **Configuração 285 –** CAP 281 e Memória 110;
- **Configuração 287 –** CAP 281 e Memória 130;

Os dados para esta figura estão estruturados em blocos, os quais correspondem às seguintes configurações iniciais: (a) 281, (b) 283, (c), 285 e (d) 287. Cada uma dessas configurações iniciais tem as configurações 239, 241, 243 e 245, respectivamente, como alvo.

```
@relation Arff_85

@attribute CAP_INICIAL {218, 225, 232, 239, 246, 253, 260, 267, 274, 281, 288,
295, 302, 309, 316, 323, 330, 337}
@attribute MEM_INICIAL {70, 80, 90, 100, 110, 120, 130}
@attribute CAP_ALVO {218, 225, 232, 239, 246, 253, 260, 267, 274, 281, 288, 295,
302, 309, 316, 323, 330, 337}
@attribute MEM_ALVO {70, 80, 90, 100, 110, 120, 130}
@attribute class  {0, 1}


@data

281,  70, 239,  70, 0
281,  70, 239,  90, 0
281,  70, 239, 110, 1
281,  70, 239, 130, 0

281,  90, 239,  70, 0
281,  90, 239,  90, 0
281,  90, 239, 110, 1
281,  90, 239, 130, 0

281, 110, 239,  70, 0
281, 110, 239,  90, 0
281, 110, 239, 110, 1
281, 110, 239, 130, 0

281, 130, 239,  70, 0
281, 130, 239,  90, 0
281, 130, 239, 110, 1
281, 130, 239, 130, 0
```

(a) (b) (c) (d)

Figura 9 – Exemplo de conteúdo do arquivo arff para a mineração

5.1.2.3. Execução do Algoritmo

Cada arquivo arff preparado é carregado no ambiente Weka e, sobre eles, é executado o algoritmo J48, utilizando o método de validação cruzada. Os modelos preditivos produzidos classificam situações que demandam, ou não, reconfiguração. Para os experimentos efetuados, os modelos fizeram uso do atributo CAP_Alvo como raiz, derivando um ou mais atributos até atingir o atributo preditivo. A Figura 10 ilustra as diferentes situações produzidas pelo classificador, quais sejam:

(a) Para um dado valor de CAP alvo (232), é verificado um ou mais valores de CAP inicial (246, 253, 260, 267, 274 e 281);

(b) Para um dado valor de CAP alvo (239), são verificados os diferentes valores de memória alvo (70, 80, 90, 100, 110, 120 e 130)

(c) Para um dado valor de CAP alvo (281), são verificados diferentes valores de CAP inicial (239) e, para um dado valor de CAP inicial, diferentes valores de memória inicial (70, 80, 90, 100, 110, 120 e 130);

(d) Para um dado valor de CAP alvo (246), são verificados diferentes valores de CAP inicial (253) e, para um dado valor de CAP inicial, diferentes valores de memória alvo (70, 80, 90, 100, 110, 120e 130); e

(e) Para um dado valor de CAP alvo (302) são verificados os diferentes valores de memória alvo (100) e, para um dado valor de memória alvo, diferentes valores de CAP inicial (218, 225, 232, 239, 246 e 253).

(a)	(b)	(c)	(d)	(e)
CAP_ALVO = 232	CAP_ALVO = 239	CAP_ALVO = 281	CAP_ALVO = 246	CAP_ALVO = 302
| CAP_INICIAL = 246: 1	| MEM_ALVO = 70: 0	| CAP_INIC IAL= 239	| CAP_INICIAL = 253	| MEM_ALVO = 100
| CAP_INICIAL = 253: 1	| MEM_ALVO = 80: 0	| | MEM_INIC = 70: 1	| | MEM_ALVO = 70: 0	| | CAP_INICIAL = 218: 0
| CAP_INICIAL = 260: 0	| MEM_ALVO = 90: 0	| | MEM_INIC = 80: 1	| | MEM_ALVO = 80: 0	| | CAP_INICIAL = 225: 1
| CAP_INICIAL = 267: 1	| MEM_ALVO = 100: 0	| | MEM_INIC = 90: 1	| | MEM_ALVO = 90: 1	| | CAP_INICIAL = 232: 1
| CAP_INICIAL = 274: 0	| MEM_ALVO = 110: 1	| | MEM_INIC = 100: 1	| | MEM_ALVO = 100: 0	| | CAP_INICIAL = 239: 1
| CAP_INICIAL = 281: 0	| MEM_ALVO = 120: 0	| | MEM_INIC = 110: 0	| | MEM_ALVO = 110: 1	| | CAP_INICIAL = 246: 1
	| MEM_ALVO = 130: 0	| | MEM_INIC = 120: 1	| | MEM_ALVO = 120: 0	| | CAP_INICIAL = 253: 1
		| | MEM_INIC = 130: 1	| | MEM_ALVO = 130: 1	

Figura 10 – Exemplos de modelos preditivos

A interpretação dos resultados é dada como segue. De acordo com Figura 10a, conclui-se que, para quaisquer configurações de memória inicial dos CAPs iniciais 246, 253 e 267, a reconfiguração para o CAP 232, independente da configuração de memória alvo, é recomendada, enquanto a reconfiguração para o CAP 232 não é recomendada quando o CAP inicial for 260, 274 e 281. Com relação à Figura 10b, a reconfiguração para o CAP alvo 239 é recomendada, para quaisquer configurações de CAP e memória inicial, quando a memória alvo for 110, somente. A Figura 10c indica que a reconfiguração do CAP inicial 239 para o CAP alvo 281, com quaisquer valores de memória, é recomendada, exceto quando a memória inicial for 110. Pela Figura 10d é possível constatar que, partindo-se do CAP inicial 253, é recomendado reconfigurar para o CAP alvo 246, com quaisquer combinações de memória inicial, quando a memória alvo for 90, 110 e 130. Por fim, a Figura 10e mostra que, para quaisquer valores de memória dos CAPs iniciais 225, 232, 239, 246 e 253, é recomendado reconfigurar para o CAP 302 se a memória alvo for 100. O resultado completo para o arquivo Arff_85 é apresentado no Anexo A.

5.1.2.4. Credibilidade da Solução

A matriz de confusão referente a cada modelo gerado está reportada na Tabela 17. A acurácia de cada modelo e, em especial, a acurácia para a predição positiva, é mostrada na Tabela 18. Pode-se constatar na Tabela 18 que há uma redução da acurácia com um número maior de instâncias tratadas. Isso parece estar mais relacionado ao número de combinações de CAP que cada arquivo preparado contém, do que o seu número de instâncias. Por outro lado, a acurácia específica para as predições positivas é praticamente a mesma do que a acurácia do modelo, não parecendo haver algum tipo de distorção ou tendência nas árvores produzidas que desconsiderem, significativamente, mudanças benéficas de reconfiguração. Contudo, testes mais exaustivos atuando, principalmente, no critério do benefício em reconfigurar, merecem ser feitos.

Tabela 17 – Matriz de confusão para os modelos gerados

Atributo Preditivo		Arff_70		Arff_85		Arff_90		Arff_100	
		Valor Predito		Valor Predito		Valor Predito		Valor Predito	
		1	0	1	0	1	0	1	0
Valor Real	1	1840	215	6700	1169	8425	2365	16453	4855
	0	262	1652	1377	6630	2836	7983	5767	14134

Tabela 18 – Acurácia dos modelos gerados

Modelo / Acurácia	Arff_70	Aff_85	Arff_90	Arff_100
Acurácia	87.9819 %	83.9632 %	75.9313 %	74.2241 %
Acurácia das predições benéficas	87,5356 %	82,9515 %	74,8157 %	74,0459 %

5.2. Apresentação e Efetivação da Reconfiguração

O formato apresentado na Figura 10 não é confortável para que o subsistema de reconfiguração desenvolvido por [ROS07] o interprete, pois o mesmo precisa de uma estrutura de dados suficientemente simples e eficiente para não onerar sua execução. Assim, é objeto, para integração dessas pesquisas, definir um protocolo de comunicação adequado, o qual pode ser estabelecido, por exemplo, em formato XML. Para tanto, faz-se necessário enriquecer os modelos de gerados, etapa (h) da Figura 6, de maneira que o subsistema de reconfiguração possa identificar, rapidamente, uma situação de uma dada máquina Xen e configurá-la de acordo com os parâmetros sugeridos.

O enriquecimento dos modelos preditivos constitui-se de instruções em alto nível para a reconfiguração do ambiente. Cada modelo preditivo enriquecido segue o modelo Evento-Condição-Ação. Os eventos são capturados pelo subsistema, descrevendo quais situações iniciam procedimentos para reconfiguração. As condições servem para selecionar o modelo preditivo adequado, já existente, e explorá-lo. Essas aparecem como um conjunto de parâmetros que representam as condições que possam sofrer alterações, os quais irão definir em que configuração atual a máquina Xen se encontra. As ações definem quais parâmetros de reconfiguração devem ser modificados, e como isso deve ser feito. Essas assumem o conjunto de configurações alvo disponível para a configuração inicial selecionada. Caso exista mais de uma opção, o subsistema avalia se tais configurações são convenientes, considerando políticas de SLA que possam estar definidas, e seleciona a que entender ser a mais conveniente.

Para melhor interpretar os resultados dos modelos preditivos gerados, os mesmos são organizados no formato apresentado pela Tabela 19. Esta contém em todo, ou em parte, os resultados mostrados na Figura 10 e é estruturada da seguinte maneira: para uma dada configuração inicial são apresentadas apenas as configurações alvo benéficas para a reconfiguração. Esses resultados de configuração alvo são enriquecidos de modo que apenas os conteúdos de CAP e memória que se diferem da configuração inicial são indicados para a reconfiguração. Vale ressaltar que essa tabela não apresenta de forma exaustiva todas as possibilidades de configuração de memória inicial cuja reconfiguração é benéfica, sugeridas pelo modelo. Assim, o conteúdo da Tabela 19 corresponde a:

- A primeira ocorrência diz respeito ao resultado do modelo apresentado na Figura 10c. Essa mostra CAP inicial 239 (10, 10, 20, 45) e, para fins de exemplificação, memória inicial 90. Com essa configuração inicial, o modelo indica que há benefício em configurar para o CAP 281 (10, 15, 25, 35) para quaisquer configurações de memória. As transições para estas reconfigurações estão apresentadas na configuração alvo e os parâmetros que devem ser alterados, nas instruções enriquecidas; e
- A segunda ocorrência diz respeito ao resultado do modelo apresentado na Figura 10d. Essa mostra o CAP inicial 253 (10, 15, 20, 40) e, para fins de exemplificação, memória inicial 130. Com essa configuração inicial, o modelo indica que há benefício em reconfigurar para o CAP 246 (10, 10, 25, 40) quando a memória alvo for 90, 110 e 130. Assim, em configuração alvo, é

apresentada apenas as transições que satisfaçam essa regra e, em instruções enriquecidas, os parâmetros que devem ser alterados.

Tabela 19 – Exemplos de instruções enriquecidas

Configuração Inicial					Confiuguração Alvo								Instruções Enriquecidas							
					CAP				MEM				VM1		VM2		VM3		VM4	
	VM 1	VM 2	VM 3	VM 4	VM 1	VM 2	VM 3	VM 4	VM 1	VM 2	VM 3	VM 4	CAP	MEM	CAP	MEM	CAP	MEM	CAP	MEM
CAP	10	10	20	45	10	15	25	35	70	70	70	70		70	15		25		35	70
									80	70	70	60		80	15		25		35	60
									90	70	70	50			15		25		35	
MEM	90	70	70	50					100	70	70	40		100	15		25		35	40
									110	70	60	40		110	15		25	60	35	40
									120	70	50	40		120	15		25	50	35	40
									130	70	40	40		130	15		25	40	35	40
CAP	10	15	20	40	10	10	25	40	90	70	70	50		70	10		25	70		50
									110	70	60	40		110	10		25	60		
MEM	130	70	40	40					130	70	40	40			10		25	40		

O subsistema de reconfiguração avalia as instruções sugeridas e, dentre as que não violam regras de SLA, elege a que considerar mais apropriada. É importante ressaltar que as combinações de configurações de CAP e memória não são permutadas entre as VMs, de modo que as diferentes distribuições não estão reportadas nos modelos preditivos. Como conseqüência, o subsistema de reconfiguração torna-se responsável por avaliar a distribuição dessas combinações e aplicar a configuração adequada, pois o custo computacional necessário é menor do que o custo que o subsistema teria em percorrer todas as combinações de instruções, e selecionar a mais adequada.

5.3. Considerações do Capítulo

Neste capítulo foram apresentadas as etapas referentes à descoberta de padrões para a reconfiguração do processo de KDD proposto. Foi relatado como o dados contidos no DW são preparados para que possam ser utilizados pelo algoritmo de mineração. O algoritmo escolhido foi o J48 do ambiente Weka. A preparação se deu em duas etapas: a primeira para o atributo preditivo e a segunda para os atributos explanatórios. A preparação foi conduzida de maneira que o resultado do algoritmo de mineração indicasse, dada uma configuração vigente, um conjunto de novas configurações a serem aplicadas para que haja ganho de desempenho. Cada percentual de CPU definido no planejamento gerou um arquivo distinto para a mineração. Em virtude da preparação definida, os modelos preditivos gerados para tais

arquivos apresentaram uma acurácia satisfatória. Foi apresentado com esses modelos são enriquecidos para que o subsistema de reconfiguração possa interpretá-los e, de maneira rápida e eficiente, reconfigurar uma máquina Xen de acordo com os parâmetros sugeridos.

6. TRABALHOS RELACIONADOS

Nesse capítulo são apresentados alguns trabalhos relacionados ao tema desta pesquisa. Esses trabalhos são relatados a seguir. Na seção subseqüente é feito um estudo comparativo entre esses trabalhos e a proposta endereçada por esta pesquisa.

O trabalho apresentado por [PAR06] relata um projeto para auxiliar na reconfiguração de ambientes de aplicações multi-camadas, em tempo de execução, sem que tal reconfiguração viole algum SLO (*service level objective*). Para tanto, é medido o desempenho de tais aplicações com o *benchmark* RUBiS [RUB07], o qual retorna 220 métricas. Dessas 220 métricas, são selecionadas apenas 12, as quais apresentam alguma relação com os SLOs definidos. A execução desse *benchmark* é efetuada sobre diferentes configurações de ambiente. Com os resultados para os diferentes cenários, é elaborado o conjunto de treino para o classificador. Enquanto os atributos explanatórios correspondem ao ambiente analisado e às métricas utilizadas e seus respectivos resultados, o atributo preditivo diz respeito ao grau de satisfação que tais resultados apresentam em relação ao SLO. Assim, com a utilização dos algoritmos de classificação C4.5, LogitBoost e Redes Bayesianas, foram produzidos modelos preditivos que indicam situações que representam gargalos. Ao identificar alguma dessas configurações-gargalos, o sistema pode ser reconfigurado para outras que não o sejam.

Em [CAL07] é apresentado um método analítico para medir desempenho de aplicações multi-camadas. A estratégia utilizada é, a partir de diversas combinações de parâmetros de configurações de ambientes, executar o *benchmark* RUBiS e RUBBoS [RUO07] para coletar métricas. Assim como no trabalho anterior, este objetiva reconfigurar o ambiente, aplicando as configurações que apresentam melhor desempenho.

[UDU07] propõe a execução do *benchmark* RUBiS sobre diferentes configurações de ambiente para extrair métricas de desempenho. O objetivo é, a partir dos resultados de cada métrica, definir políticas de SLA. Para tanto, são utilizados os algoritmos C4.5 e Random Tree, onde os atributos explanatórios são as métricas e seus valores, e o atributo preditivo é um atributo binário. O modelo preditivo gerado indica quais valores de métricas estão de acordo com políticas de SLA.

O trabalho apresentado por [CHE07] propõe métodos analíticos para capturar o relacionamento entre métricas de desempenho obtidas dos *benchmarks* TPC-W [TPC07] e RUBiS para ajustar políticas de SLA, definindo novos SLOs, em ambientes de *data centers*

que se beneficiam com paravirtualizadores, como o Xen. A idéia é capturar métricas de desempenho e, com métodos analíticos, analisar os ambientes para que, a partir de seu comportamento, possam ser definidos novos SLOs para manter atualizadas as políticas de SLA.

Em [CUN07] é sugerida a reconfiguração dinâmica para melhorar o desempenho de ambientes virtualizados. A abordagem é focalizada em capturar o desempenho de sistemas com diferentes configurações e, a partir de métodos analíticos que comparam um dado resultado de desempenho com políticas de SLA, definir uma nova configuração que apresente melhora de desempenho para o ambiente.

6.1. Considerações do Capítulo

A Tabela 20 mostra uma comparação entre os trabalhos relacionados, em relação aos aspectos que esta pesquisa endereça (apresentado na última coluna dessa tabela). Tais aspectos são os seguintes:

- **Objetivo –** o que o trabalho se propõe a testar: reconfiguração do ambiente analisado ou definição de políticas de SLA. Diferentes objetivos implicam em distintos requisitos a serem atendidos;
- **Ambiente Analisado –** qual tipo de ambiente é alvo: ambientes virtualizados como, por exemplo, utilizando o Xen, ou ambientes multi-camadas, com aplicações cliente-servidor;
- **Métricas –** quais as fontes para obtenção das métricas: podem ser obtidas de distintos *benchmarks*;
- **Modelo Analítico de Dados –** se a proposta organiza os dados em um DW para facilitar sua manipulação;
- **Preparação de Dados** – se a proposta emprega algum procedimento para a preparação a fim de melhorar a qualidade da solução; e
- **Análise das Métricas** – qual a abordagem adotada para análise das métricas: mineração de dados ou métodos analíticos;

Não fica claro, nos trabalhos acima, se é adotado ou proposto um processo completo de KDD, em especial nas etapas de preparação de dados e seu apropriado armazenamento em

um DW. A preparação de dados é percebida apenas em [PAR06]. Por outro lado, os trabalhos que utilizam mineração de dados, [PAR06] e [UDU07], utilizam o algoritmo J48 do Weka para atender a seus objetivos. Como pode ser observado na Tabela 20, esta proposta é a única que endereça a reconfiguração de ambientes virtualizados com análise das métricas através de técnicas de mineração de dados. Além disso, o uso de um DW e o emprego rigoroso de técnicas de preparação de dados aumenta a qualidade desta solução.

Tabela 20 – Comparação entre trabalhos relacionados

Trabalho / Aspecto	**[PAR06]**	**[CAL07]**	**[UDU07]**	**[CHE07]**	**[CUN07]**	**Proposta deste Trabalho**
Objetivo	Reconfiguração do ambiente	Reconfiguração do ambiente	Políticas de SLA	Políticas de SLA	Reconfiguração do ambiente	Reconfiguração do ambiente
Ambiente Analisado	Multi-camadas	Multi-camadas	Muli-camadas	Virtualizado	Virtualizado	Virtualizado
Métricas	*Benchmark* RUBiS	*Benchmarks* RUBiS e RUBBoS	*Benchmark* RUBiS	*Benchmarks* TPC-W e RUBiS	Não deixa claro como são obtidas	*Benchmark* Unixbench
Modelo de Dados	Não utiliza	Não utiliza	Não utiliza	Não utiliza	Não utiliza	*Data Warehouse*
Preparação de Dados	Aplica regras para o definir o conjunto de treino	Não endereça	Não deixa claro	Não endereça	Não endereça	São aplicadas as etapas de preparação sugeridas por [TAN06]
Análise das Métricas	Mineração de Dados	Método Analítico	Mineração de Dados	Método Analítico	Método Analítico	Mineração de Dados

7. CONCLUSÕES

Este trabalho apresentou o Xen, um paravirtualizador que permite a execução simultânea de diversas VMs, cada uma com seu próprio sistema operacional. Para tirar proveio de sua estrutura, foi apresentada uma maneira ideal para realocação dos recursos consumidos pelas VMs, a fim de que seja obtido um desempenho mais satisfatório, onde a abordagem adotada é a de execuções de *benchmarks* para capturar métricas de desempenho das VMs. Para auxiliar a reconfiguração de parâmetros, foi construído um processo completo de KDD para capturar dados de execuções de *benchmarks*, organizá-los em um DW e aplicar técnicas de mineração para sugerir novos parâmetros. Essa pesquisa está inserida em um projeto maior, onde a efetivação da reconfiguração é objeto de outra pesquisa complementar, que propõe um subsistema para interpretar os parâmetros sugeridos e realizar a reconfiguração.

Para o *data warehousing* do processo de KDD, foi apresentado um planejamento para execuções de *benchmarks*, os quais englobam diferentes configurações que possam ocorrer para o ambiente analisado. O modelo analítico desenvolvido para armazenar as métricas de *benchmarks* foi modelado para comportar quaisquer variações de configuração de ambiente em que esses *benchmarks* sejam executados, bem como para comportar métricas de distintos *benchmarks*. A organização desses dados em um DW permite que, sobre os mesmos, sejam efetuadas consultas analíticas para obter detalhamentos em diferentes perspectivas.

A etapa de descoberta de padrões para a reconfiguração, do processo de KDD, apresentou como os dados contidos no DW podem ser utilizados por técnicas de mineração de dados, a fim de inferir sobre novos parâmetros de reconfiguração para o Xen, onde o algoritmo utilizado foi o J48 do Weka. Para que esse algoritmo pudesse produzir resultados que sugerissem, seletivamente, novos parâmetros de configuração para uma dada máquina Xen vigente, foi feito uso rigoroso de técnicas de preparação de dados. A preparação dos dados se deu em duas etapas: a primeira, para definir o atributo preditivo, e a segunda, para preparar os atributos explanatórios. Para a preparação, o critério adotado foi de comparar o desempenho entre uma dada configuração inicial, com possíveis configurações alvo. Assim, puderam ser gerados modelos preditivos que indicassem conjuntos de parâmetros que apresentam melhora de desempenho para uma dada configuração do Xen.

Os modelos preditivos produzidos não são confortáveis para que o subsistema de reconfiguração os possa interpretar. Nesse sentido, apresentou-se como os mesmos podem ser enriquecidos com instruções em alto nível que indiquem quais parâmetros de configuração merecem ser alterados, onde tais instruções são de simples interpretação e eficientes para a reconfiguração.

A proposta desta pesquisa mostrou-se abrangente por propor um processo completo de KDD, endereçando todas as etapas inerentes a tal processo. Embora cada uma dessas etapas sejam suficientemente amplas para serem individualmente tratadas, entendeu-se que endereçar o processo como um todo é fundamental para atender aos objetivos propostos. Não foram encontrados trabalhos com essa abordagem, para esse tipo de problema, ou seja, que utilizasse um processo de KDD, empregando técnicas de mineração de dados para a reconfiguração de ambientes virtualizados. Além disso, entendeu-se que o uso de um DW e o emprego rigoroso de técnicas de preparação de dados corroboraram para a solução como um todo.

7.1. Trabalhos Futuros

Por se tratar de um tema abrangente, endereçando todas as etapas que compõem o processo de KDD, essa proposta pode ser beneficiada se sejam agregadas novas características. Assim, no que refere-se ao *data warehousing* de métricas de *benchmarks*, as seguintes abordagens seriam benéficas:

- Para o teste da Fonte de Dados, o planejamento poderia considerar diferentes tipos de hardware de modo que, nas etapas subseqüentes, pudessem ser endereçadas a migração de VMs entre máquinas;
- Para o planejamento definido, podem ser executadas mais vezes o mesmo *benchmark*, bem como ser executados outros *benchmarks*. Assim, o volume de resultados de desempenho poderia colaborar para a futura produção de modelos preditivos; e
- O processo de ETC dos resultados de *benchmarks* para o DW poderia ser aprimorado caso recebesse um nível maior de automação.

No contexto de descoberta de padrões para a reconfiguração, embora os modelos preditivos tenham apresentado uma acurácia satisfatória, os mesmos poderiam ser beneficiados se adotassem os seguintes critérios:

- A preparação do atributo preditivo poderia ser abordado de maneira um pouco distinta: hoje a preparação é feita através da comparação entre uma configuração alvo e uma configuração inicial, onde o valor da diferença entre o desempenho dessas duas configurações determina o atributo preditivo. Existe, entretanto, diferenças mínimas, cujo custo para a reconfiguração pode não valer a pena. Nesse sentido, caso fosse adotada uma margem para essa diferença na determinação do atributo preditivo, a acurácia dos modelos poderia ser melhorada; e
- A adoção de políticas de SLA é, atualmente, verificada pelo subsistema no momento da reconfiguração. Essas poderiam ser analisadas e preparadas para serem endereçadas também na mineração de dados.

REFERÊNCIAS

[BAR03] BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. 2003. Xen and the art of virtualization. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES (SOSP'03), 19., 2003, New York. **Proceedings…** New York: ACM Press, Out. 2003, p. 164-177.

[BEC06] BECKER, K.; RUIZ, D.; CUNHA, V.; NOVELLO, T.; SOUZA, F. SPDW : a software development process performance data warehousing environment. In: Annual IEEE/NASA Software Engineering Workshop (SEW'06), 30., 2006, Betheda, MD. **Proceedings**… Los Alamitos: IEEE Computer Society, Abr. 2006. p. 107-118.

[CAL07] PU, C,; SAHAI, A.; PAREKH, J.; BAE, J.; CHA, Y.; GARCIA, T.; IRANI, D.; LEE, J.; LIN, Q. An observation-based approach to performance characterization of distributed n-tier applications. In: INTERNATIONAL SYMPOSIUM ON WORKLOAD CHARACTERIZATION (IISWC´07), 1., 2007, Boston. **Proceedings…** Los Alamitos, IEEE Computer Society, Set 2007, p. 161-170.

[CHE07] CHEN, Y., IYER, S., LIU, X., MOLOJICIC, D., SAHAY, A. Translating Service Level Objectives to System Level Thresholds. In: INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING (ICAC'07), 4., 2007, Washington. **Proceedings…** Washington: IEEE Computer Society, Jun. 2007, p. 3-13.

[CLA05] CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J. G.; JUL, E.; LIMPACH, C.; PRATT, I.; WARFIELD, A. Live migration of virtual machines. In CONFERENCE ON SYMPOSIUM ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION, 2., 2005, Berkeley. **Proceedings…** Berkeley: ACM/USENIX, Maio 2005, p. 273-286.

[CUN07] CUNHA, I.; ALMEIDA, J.; ALMEIDA, V.; SANTOS, M. Self-adaptative capacity management for multi-tier virtualized environments. In: INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM'07), 10., 2007, Munich. **Proceedings…** Los Alamitos: IFIP/IEEE Computer Society, Maio 2007, p. 129-138.

[FAY96] FAYYAD, U.; PIATETSKY-SHAPIRO G.; SMYTH P. The KDD process for extracting useful knowledge from volumes of data. **Communications of the ACM**, New York, v. 39, n. 11, p.27-34, Nov. 1996.

[HAN01] HAN, J.; KAMBER, M. **Data mining**: concepts and techniques. San Francisco: Morgan Kaufmann, 2001. 550 p.

[IOZ07] IOZONE**.** Disponível em: < http://www.iozone.org>. Acesso em: 10 de maio 2007.

[KIM98] KIMBALL, R.; ROSS, M. **The data warehouse lifecycle toolkit :** expert methods for designing, developing, and deploying**.** São Paulo: Campus, 1998. 520 p.

[LIR07] LI, T.; RUAN, D. An extended process model of knowledge discovery in databases. **Enterprise Information Management**, Bingley, UK, v. 20, n. 2, p.169-177, 2007.

[MCV96] McVOY, L. W.; STAELIN, C. Lmbench: portable tools for performance analysis. In: USENIX ANNUAL TECHNICAL CONFERENCE**,** 1996, San Diego. **Proceedings…** Berkeley: USENIX Association, Jan. 1996, p. 279-294.

[MEN05] MENON, A.; SANTOS, J.; TURNER, Y.; JANAKIRAMAN, G.; ZWAENEPOEL, W. Diagnosing performance overheads in the virtual machine environment. In: INTERNATIONAL CONFERENCE ON VIRTUAL ENVIRONMENTS (VEE´05), 1., 2005, New York. **Proceedings…** New York: ACM/USENIX, Jun. 2007, p. 13-23.

[MER06] MERGEN, M. F.; UHLIG, V.; KRIEGER, O.; XENIDIS, J. 2006. Virtualization for high-performance computing. SIGOPS. **Operating System. Review**, New York, v. 40, n. 2, p. 8-11, Abr. 2006.

[PAR06] PAREKH, J.; JUNG, G.; SWINT, G.; PU, C.; SAHAI, A. Comparison of performance analysis approaches for bottleneck detection in multi-tier enterprise applications. In: INTERNATIONAL WORKSHOP ON QUALITY OF SERVICE (IWqOS'06), 14., 2006, New Haven. **Proceedings…** Los Alamitos: IEEE Computer Society, Jun. 2006, p. 302-311.

[QUI96] QUINLAN, J. R. **C4.5:** programs for machine learning. São Francisco: Morgan Kaufmann, 1996. 302 p.

[ROS05] ROSEMBLUM, M.; GARFINKEL, T. Virtual machine monitors: current technology and future trends. **Computer**, Los Alamitos, v. 38, n. 5, p. 39-47, Maio, 2005.

[ROS07] ROSSI, F. **Alocação dinâmica de recursos no Xen.** 64 f. 2007. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Informática – PUCRS, Porto Alegre, 2008.

[RUB07] RUBIS: Rice university bidding system. Disponível em: <http://rubis.objectweb.org/>. Acesso em 15 de maio 2007.

[RUO07] RUBBOS: Bulletin board benchmark. Disponível em: <http://jmob.objectweb.org/rubbos.html>. Acesso em 8 de dez. 2007.

[SIL07] SILVEIRA, P. **Processo de ETC orientado a serviço para um ambiente de gestão de qualidade de software.** 162 f. 2007. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Informática – PUCRS, Porto Alegre, 2007.

[TAN06] TAN, P-N.; STEINBACH, M., KUMAR, V. **Introduction to data mining**. Boston: Addison Wesley, 2006. 769 p.

[TPC07] **TPC-W**: Transaction processing performance council Web. Disponível em: <http://www.tpc.org/tpcw/>. Acesso em 10 de dez. 2007.

[UDU07] UDUPI, B.; SAHAI, A.; SINGHAL, S. A classification-based approach to policy refinement. In: SYMPOSIUM ON INTEGRATED MANAGEMENT (IFIP/IM'07), 10., 2007, Munich. **Proceedings…** Los Alamitos: IEEE Computer Society, May 2007, p. 785-788.

[UNI07] UNIXBENCH. Disponível em: < http://www.tux.org/pub/tux/niemi/unixbench/>. Acesso em 15 de maio 2007.

[WEK07] WEKA: Waikato environment for knowledge analysis. Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/>. Acesso em 15 de maio 2007.

[WIT05] WITTEN, I.; FRANK, E. **Data mining**: pratical machine learning tools and techniques. San Francisco: Morgan Kaufmann, 2005. 525 p.

ANEXO A – MODELO PREDITIVO PARA O ARQUIVO ARFF_85

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     RESULT
Instances:    15876
Attributes:   5
              CAP_INICIAL
              MEM_INICIAL
              CAP_ALVO
              MEM_ALVO
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

CAP_ALVO = 218
|   MEM_ALVO = 70: 1 (126.0/3.0)
|   MEM_ALVO = 80: 1 (126.0/9.0)
|   MEM_ALVO = 90: 1 (126.0/8.0)
|   MEM_ALVO = 100: 1 (126.0/6.0)
|   MEM_ALVO = 110: 1 (126.0/5.0)
|   MEM_ALVO = 120: 0 (126.0/1.0)
|   MEM_ALVO = 130: 1 (126.0/9.0)
CAP_ALVO = 225
|   MEM_ALVO = 70: 1 (126.0/2.0)
|   MEM_ALVO = 80
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/3.0)
|   |   CAP_INICIAL = 232: 0 (7.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0/1.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/1.0)
|   |   CAP_INICIAL = 309: 1 (7.0/2.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/2.0)
|   MEM_ALVO = 90: 0 (126.0/6.0)
|   MEM_ALVO = 100: 0 (126.0/7.0)
|   MEM_ALVO = 110: 1 (126.0/4.0)
|   MEM_ALVO = 120: 0 (126.0/9.0)
|   MEM_ALVO = 130: 0 (126.0/15.0)
CAP_ALVO = 232
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 232
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 0 (7.0/3.0)
|   |   MEM_INICIAL = 90: 1 (7.0/3.0)
|   |   MEM_INICIAL = 100: 0 (7.0/2.0)
|   |   MEM_INICIAL = 110: 1 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 239
```

```
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 246: 1 (49.0)
|   CAP_INICIAL = 253: 1 (49.0)
|   CAP_INICIAL = 260: 0 (49.0/5.0)
|   CAP_INICIAL = 267: 1 (49.0)
|   CAP_INICIAL = 274: 0 (49.0/4.0)
|   CAP_INICIAL = 281: 0 (49.0)
|   CAP_INICIAL = 288
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 295: 1 (49.0/6.0)
|   CAP_INICIAL = 302
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 309
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0/2.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 316
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 323
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 330
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 239
|   MEM_ALVO = 70: 0 (126.0/3.0)
|   MEM_ALVO = 80: 0 (126.0/6.0)
|   MEM_ALVO = 90: 0 (126.0/9.0)
|   MEM_ALVO = 100: 0 (126.0/12.0)
|   MEM_ALVO = 110: 1 (126.0/1.0)
|   MEM_ALVO = 120: 0 (126.0/2.0)
|   MEM_ALVO = 130: 0 (126.0/12.0)
CAP_ALVO = 246
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
```

```
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0/1.0)
|   |   MEM_INICIAL = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 232: 0 (49.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 246
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/2.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0/3.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 1 (7.0/3.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
|   CAP_INICIAL = 253
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 1 (7.0)
|   |   MEM_ALVO = 100: 0 (7.0/3.0)
|   |   MEM_ALVO = 110: 1 (7.0/2.0)
|   |   MEM_ALVO = 120: 0 (7.0/1.0)
|   |   MEM_ALVO = 130: 1 (7.0/2.0)
|   CAP_INICIAL = 260: 0 (49.0)
|   CAP_INICIAL = 267
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0/1.0)
|   |   MEM_ALVO = 90: 1 (7.0)
|   |   MEM_ALVO = 100: 1 (7.0/3.0)
|   |   MEM_ALVO = 110: 1 (7.0/1.0)
|   |   MEM_ALVO = 120: 0 (7.0/2.0)
|   |   MEM_ALVO = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 274: 0 (49.0)
|   CAP_INICIAL = 281: 0 (49.0)
|   CAP_INICIAL = 288: 0 (49.0/15.0)
|   CAP_INICIAL = 295
|   |   MEM_INICIAL = 70: 1 (7.0/2.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0/3.0)
|   |   MEM_INICIAL = 100: 1 (7.0/3.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 302: 0 (49.0/6.0)
|   CAP_INICIAL = 309
|   |   MEM_INICIAL = 70: 0 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/3.0)
|   |   MEM_INICIAL = 90: 1 (7.0/2.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 316: 0 (49.0/6.0)
|   CAP_INICIAL = 323: 0 (49.0/13.0)
|   CAP_INICIAL = 330: 0 (49.0/7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0/3.0)
|   |   MEM_INICIAL = 80: 0 (7.0/3.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/2.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 253
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
```

```
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 232: 0 (49.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 246
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0/2.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 1 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
|   CAP_INICIAL = 253
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0/3.0)
|   |   MEM_ALVO = 90: 1 (7.0/1.0)
|   |   MEM_ALVO = 100: 0 (7.0/3.0)
|   |   MEM_ALVO = 110: 0 (7.0/1.0)
|   |   MEM_ALVO = 120: 0 (7.0/1.0)
|   |   MEM_ALVO = 130: 1 (7.0/2.0)
|   CAP_INICIAL = 260: 0 (49.0)
|   CAP_INICIAL = 267
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/2.0)
|   |   MEM_INICIAL = 110: 0 (7.0/2.0)
|   |   MEM_INICIAL = 120: 1 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
|   CAP_INICIAL = 274: 0 (49.0)
|   CAP_INICIAL = 281: 0 (49.0)
|   CAP_INICIAL = 288
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0/2.0)
|   |   MEM_INICIAL = 90: 0 (7.0/2.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0/2.0)
|   CAP_INICIAL = 295
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0/3.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 302
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 309
|   |   MEM_INICIAL = 70: 0 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/3.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 316: 0 (49.0/4.0)
|   CAP_INICIAL = 323: 0 (49.0/9.0)
|   CAP_INICIAL = 330: 0 (49.0/8.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 0 (7.0/2.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 260
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
```

```
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 232: 1 (49.0/5.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 246: 1 (49.0)
|   CAP_INICIAL = 253: 1 (49.0)
|   CAP_INICIAL = 260
|   |   MEM_INICIAL = 70: 1 (7.0/3.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 1 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
|   CAP_INICIAL = 267: 1 (49.0)
|   CAP_INICIAL = 274
|   |   MEM_INICIAL = 70: 1 (7.0/2.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/2.0)
|   |   MEM_INICIAL = 110: 1 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
|   CAP_INICIAL = 281: 0 (49.0/6.0)
|   CAP_INICIAL = 288
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0/3.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 295: 1 (49.0/2.0)
|   CAP_INICIAL = 302
|   |   MEM_INICIAL = 70: 0 (7.0/1.0)
|   |   MEM_INICIAL = 80: 0 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0/3.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 309: 1 (49.0/6.0)
|   CAP_INICIAL = 316
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0/2.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 323
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 330
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
```

```
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 267
|   MEM_ALVO = 70: 0 (126.0/14.0)
|   MEM_ALVO = 80: 0 (126.0/19.0)
|   MEM_ALVO = 90: 0 (126.0/25.0)
|   MEM_ALVO = 100: 1 (126.0/10.0)
|   MEM_ALVO = 110: 0 (126.0/36.0)
|   MEM_ALVO = 120: 0 (126.0/25.0)
|   MEM_ALVO = 130: 0 (126.0/50.0)
CAP_ALVO = 274
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 232: 1 (49.0/6.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 246: 1 (49.0)
|   CAP_INICIAL = 253: 1 (49.0)
|   CAP_INICIAL = 260
|   |   MEM_ALVO = 70: 0 (7.0/2.0)
|   |   MEM_ALVO = 80: 1 (7.0)
|   |   MEM_ALVO = 90: 1 (7.0/1.0)
|   |   MEM_ALVO = 100: 0 (7.0/2.0)
|   |   MEM_ALVO = 110: 0 (7.0/2.0)
|   |   MEM_ALVO = 120: 1 (7.0/1.0)
|   |   MEM_ALVO = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 267: 1 (49.0)
|   CAP_INICIAL = 274
|   |   MEM_INICIAL = 70: 1 (7.0/3.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 1 (7.0/3.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0/3.0)
|   CAP_INICIAL = 281: 0 (49.0/9.0)
|   CAP_INICIAL = 288
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0/3.0)
|   |   MEM_INICIAL = 110: 0 (7.0/3.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 295: 1 (49.0/3.0)
|   CAP_INICIAL = 302
|   |   MEM_INICIAL = 70: 0 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/3.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/3.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0/3.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 309: 1 (49.0/6.0)
|   CAP_INICIAL = 316
|   |   MEM_INICIAL = 70: 0 (7.0/1.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0/3.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0/3.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 323
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
```

```
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 330
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/3.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 281
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 232: 1 (49.0/1.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 246: 1 (49.0)
|   CAP_INICIAL = 253: 1 (49.0)
|   CAP_INICIAL = 260: 1 (49.0/7.0)
|   CAP_INICIAL = 267: 1 (49.0)
|   CAP_INICIAL = 274: 1 (49.0/12.0)
|   CAP_INICIAL = 281
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/2.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/3.0)
|   |   MEM_INICIAL = 110: 0 (7.0/2.0)
|   |   MEM_INICIAL = 120: 1 (7.0/3.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 288
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 295: 1 (49.0)
|   CAP_INICIAL = 302: 1 (49.0/9.0)
|   CAP_INICIAL = 309: 1 (49.0/3.0)
|   CAP_INICIAL = 316
|   |   MEM_INICIAL = 70: 0 (7.0/3.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/2.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 323
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
```

```
|   CAP_INICIAL = 330
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0/3.0)
|   |   MEM_INICIAL = 100: 0 (7.0/3.0)
|   |   MEM_INICIAL = 110: 1 (7.0/3.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 288
|   MEM_ALVO = 70
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/3.0)
|   |   CAP_INICIAL = 232: 0 (7.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0/1.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/1.0)
|   |   CAP_INICIAL = 309: 1 (7.0/2.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/2.0)
|   MEM_ALVO = 80: 0 (126.0/8.0)
|   MEM_ALVO = 90: 0 (126.0/41.0)
|   MEM_ALVO = 100
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/2.0)
|   |   CAP_INICIAL = 232: 1 (7.0/1.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0/2.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0/3.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0)
|   |   CAP_INICIAL = 302: 0 (7.0/2.0)
|   |   CAP_INICIAL = 309: 1 (7.0/1.0)
|   |   CAP_INICIAL = 316: 0 (7.0/3.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/1.0)
|   MEM_ALVO = 110: 1 (126.0/35.0)
|   MEM_ALVO = 120: 1 (126.0/14.0)
|   MEM_ALVO = 130: 0 (126.0/36.0)
CAP_ALVO = 295
|   CAP_INICIAL = 218: 0 (49.0/6.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0/1.0)
|   |   MEM_INICIAL = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 232: 0 (49.0/5.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0/1.0)
|   |   MEM_INICIAL = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 246
|   |   MEM_ALVO = 70: 0 (7.0/1.0)
|   |   MEM_ALVO = 80: 0 (7.0/1.0)
|   |   MEM_ALVO = 90: 0 (7.0/3.0)
|   |   MEM_ALVO = 100: 0 (7.0/2.0)
|   |   MEM_ALVO = 110: 0 (7.0/3.0)
|   |   MEM_ALVO = 120: 1 (7.0/1.0)
|   |   MEM_ALVO = 130: 1 (7.0)
```

```
|   CAP_INICIAL = 253
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 0 (7.0/1.0)
|   |   MEM_ALVO = 100: 0 (7.0/1.0)
|   |   MEM_ALVO = 110: 1 (7.0/2.0)
|   |   MEM_ALVO = 120: 1 (7.0)
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 260: 0 (49.0/2.0)
|   CAP_INICIAL = 267
|   |   MEM_ALVO = 70: 0 (7.0/1.0)
|   |   MEM_ALVO = 80: 0 (7.0/1.0)
|   |   MEM_ALVO = 90: 1 (7.0/3.0)
|   |   MEM_ALVO = 100: 0 (7.0/2.0)
|   |   MEM_ALVO = 110: 1 (7.0/2.0)
|   |   MEM_ALVO = 120: 1 (7.0)
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 274: 0 (49.0/1.0)
|   CAP_INICIAL = 281: 0 (49.0)
|   CAP_INICIAL = 288: 0 (49.0/15.0)
|   CAP_INICIAL = 295
|   |   MEM_INICIAL = 70: 1 (7.0/2.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0/3.0)
|   |   MEM_INICIAL = 100: 1 (7.0/3.0)
|   |   MEM_INICIAL = 110: 1 (7.0)
|   |   MEM_INICIAL = 120: 0 (7.0/2.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 302: 0 (49.0/7.0)
|   CAP_INICIAL = 309
|   |   MEM_INICIAL = 70: 0 (7.0/2.0)
|   |   MEM_INICIAL = 80: 0 (7.0/3.0)
|   |   MEM_INICIAL = 90: 1 (7.0/2.0)
|   |   MEM_INICIAL = 100: 1 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 316: 0 (49.0/6.0)
|   CAP_INICIAL = 323
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 0 (7.0)
|   |   MEM_ALVO = 100: 0 (7.0)
|   |   MEM_ALVO = 110: 0 (7.0/3.0)
|   |   MEM_ALVO = 120: 1 (7.0/1.0)
|   |   MEM_ALVO = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 330: 0 (49.0/7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0/3.0)
|   |   MEM_INICIAL = 80: 0 (7.0/3.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 1 (7.0/2.0)
|   |   MEM_INICIAL = 110: 0 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
CAP_ALVO = 302
|   MEM_ALVO = 70: 1 (126.0/31.0)
|   MEM_ALVO = 80: 1 (126.0/41.0)
|   MEM_ALVO = 90: 1 (126.0/25.0)
|   MEM_ALVO = 100
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/2.0)
|   |   CAP_INICIAL = 232: 1 (7.0/1.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0/2.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0/3.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0)
|   |   CAP_INICIAL = 302: 0 (7.0/2.0)
|   |   CAP_INICIAL = 309: 1 (7.0/1.0)
|   |   CAP_INICIAL = 316: 0 (7.0/3.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/1.0)
|   MEM_ALVO = 110
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/2.0)
|   |   CAP_INICIAL = 232: 1 (7.0/2.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
```

```
|   |   CAP_INICIAL = 274: 0 (7.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/1.0)
|   |   CAP_INICIAL = 309: 1 (7.0/1.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/1.0)
|   MEM_ALVO = 120: 1 (126.0/35.0)
|   MEM_ALVO = 130: 0 (126.0/9.0)
CAP_ALVO = 309
|   MEM_ALVO = 70: 0 (126.0/50.0)
|   MEM_ALVO = 80: 0 (126.0/31.0)
|   MEM_ALVO = 90: 0 (126.0/19.0)
|   MEM_ALVO = 100: 0 (126.0/15.0)
|   MEM_ALVO = 110
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/3.0)
|   |   CAP_INICIAL = 232: 0 (7.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0/1.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/1.0)
|   |   CAP_INICIAL = 309: 1 (7.0/3.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/3.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/2.0)
|   MEM_ALVO = 120
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/2.0)
|   |   CAP_INICIAL = 232: 0 (7.0/1.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/1.0)
|   |   CAP_INICIAL = 309: 1 (7.0/2.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/1.0)
|   MEM_ALVO = 130: 1 (126.0/31.0)
CAP_ALVO = 316
|   MEM_ALVO = 70: 1 (126.0/25.0)
|   MEM_ALVO = 80: 1 (126.0/13.0)
|   MEM_ALVO = 90: 1 (126.0/41.0)
|   MEM_ALVO = 100: 1 (126.0/19.0)
|   MEM_ALVO = 110
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/2.0)
|   |   CAP_INICIAL = 232: 1 (7.0/2.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0/2.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0/1.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/2.0)
|   |   CAP_INICIAL = 309: 1 (7.0/1.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/1.0)
|   MEM_ALVO = 120: 1 (126.0/12.0)
|   MEM_ALVO = 130: 0 (126.0/48.0)
CAP_ALVO = 323
|   CAP_INICIAL = 218
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0)
|   |   MEM_INICIAL = 90: 0 (7.0)
```

```
|   |   MEM_INICIAL = 100: 0 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 225
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 232
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 0 (7.0)
|   |   MEM_ALVO = 100: 0 (7.0)
|   |   MEM_ALVO = 110: 0 (7.0)
|   |   MEM_ALVO = 120: 0 (7.0)
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 239
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 0 (7.0)
|   |   MEM_INICIAL = 120: 1 (7.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 246: 1 (49.0/13.0)
|   CAP_INICIAL = 253: 1 (49.0/10.0)
|   CAP_INICIAL = 260
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 0 (7.0)
|   |   MEM_ALVO = 100: 0 (7.0)
|   |   MEM_ALVO = 110: 0 (7.0)
|   |   MEM_ALVO = 120: 0 (7.0)
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 267: 1 (49.0/6.0)
|   CAP_INICIAL = 274
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 0 (7.0)
|   |   MEM_ALVO = 100: 0 (7.0)
|   |   MEM_ALVO = 110: 0 (7.0)
|   |   MEM_ALVO = 120: 0 (7.0)
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 281
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0)
|   |   MEM_ALVO = 90: 0 (7.0)
|   |   MEM_ALVO = 100: 0 (7.0)
|   |   MEM_ALVO = 110: 0 (7.0)
|   |   MEM_ALVO = 120: 0 (7.0)
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 288
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 1 (7.0/1.0)
|   CAP_INICIAL = 295: 1 (49.0/10.0)
|   CAP_INICIAL = 302
|   |   MEM_INICIAL = 70: 0 (7.0/1.0)
|   |   MEM_INICIAL = 80: 0 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 309
|   |   MEM_INICIAL = 70: 1 (7.0/3.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0/3.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
|   CAP_INICIAL = 316
|   |   MEM_ALVO = 70: 0 (7.0)
|   |   MEM_ALVO = 80: 0 (7.0/1.0)
|   |   MEM_ALVO = 90: 0 (7.0/2.0)
|   |   MEM_ALVO = 100: 0 (7.0/1.0)
|   |   MEM_ALVO = 110: 0 (7.0/2.0)
|   |   MEM_ALVO = 120: 0 (7.0/2.0)
```

```
|   |   MEM_ALVO = 130: 1 (7.0)
|   CAP_INICIAL = 323
|   |   MEM_INICIAL = 70: 1 (7.0/1.0)
|   |   MEM_INICIAL = 80: 1 (7.0/2.0)
|   |   MEM_INICIAL = 90: 1 (7.0/3.0)
|   |   MEM_INICIAL = 100: 1 (7.0/2.0)
|   |   MEM_INICIAL = 110: 0 (7.0/2.0)
|   |   MEM_INICIAL = 120: 0 (7.0/3.0)
|   |   MEM_INICIAL = 130: 0 (7.0)
|   CAP_INICIAL = 330
|   |   MEM_INICIAL = 70: 0 (7.0)
|   |   MEM_INICIAL = 80: 0 (7.0/1.0)
|   |   MEM_INICIAL = 90: 0 (7.0/1.0)
|   |   MEM_INICIAL = 100: 0 (7.0/1.0)
|   |   MEM_INICIAL = 110: 0 (7.0/1.0)
|   |   MEM_INICIAL = 120: 1 (7.0/2.0)
|   |   MEM_INICIAL = 130: 1 (7.0)
|   CAP_INICIAL = 337
|   |   MEM_INICIAL = 70: 1 (7.0)
|   |   MEM_INICIAL = 80: 1 (7.0/1.0)
|   |   MEM_INICIAL = 90: 1 (7.0)
|   |   MEM_INICIAL = 100: 1 (7.0)
|   |   MEM_INICIAL = 110: 1 (7.0/3.0)
|   |   MEM_INICIAL = 120: 0 (7.0/1.0)
|   |   MEM_INICIAL = 130: 0 (7.0/1.0)
CAP_ALVO = 330
|   MEM_ALVO = 70: 1 (126.0/10.0)
|   MEM_ALVO = 80: 1 (126.0/41.0)
|   MEM_ALVO = 90: 1 (126.0/25.0)
|   MEM_ALVO = 100: 1 (126.0/25.0)
|   MEM_ALVO = 110: 1 (126.0/31.0)
|   MEM_ALVO = 120: 1 (126.0/12.0)
|   MEM_ALVO = 130: 0 (126.0/25.0)
CAP_ALVO = 337
|   MEM_ALVO = 70: 0 (126.0/30.0)
|   MEM_ALVO = 80: 0 (126.0/41.0)
|   MEM_ALVO = 90: 0 (126.0/15.0)
|   MEM_ALVO = 100: 0 (126.0/19.0)
|   MEM_ALVO = 110: 0 (126.0/50.0)
|   MEM_ALVO = 120
|   |   CAP_INICIAL = 218: 0 (7.0/1.0)
|   |   CAP_INICIAL = 225: 1 (7.0/2.0)
|   |   CAP_INICIAL = 232: 0 (7.0)
|   |   CAP_INICIAL = 239: 1 (7.0/1.0)
|   |   CAP_INICIAL = 246: 1 (7.0)
|   |   CAP_INICIAL = 253: 1 (7.0)
|   |   CAP_INICIAL = 260: 0 (7.0)
|   |   CAP_INICIAL = 267: 1 (7.0)
|   |   CAP_INICIAL = 274: 0 (7.0)
|   |   CAP_INICIAL = 281: 0 (7.0)
|   |   CAP_INICIAL = 288: 1 (7.0/3.0)
|   |   CAP_INICIAL = 295: 1 (7.0/1.0)
|   |   CAP_INICIAL = 302: 0 (7.0/1.0)
|   |   CAP_INICIAL = 309: 1 (7.0/2.0)
|   |   CAP_INICIAL = 316: 0 (7.0/2.0)
|   |   CAP_INICIAL = 323: 1 (7.0/1.0)
|   |   CAP_INICIAL = 330: 0 (7.0/2.0)
|   |   CAP_INICIAL = 337: 1 (7.0/2.0)
|   MEM_ALVO = 130: 1 (126.0/13.0)

Number of Leaves  :                                     877

Size of the tree :                                      990


Time taken to build model: 0.11 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       13330               83.9632 %
Incorrectly Classified Instances      2546               16.0368 %
Kappa statistic                          0.6793
Mean absolute error                      0.2249
Root mean squared error                  0.351
Relative absolute error                 44.9859 %
Root relative squared error             70.2087 %
Total Number of Instances            15876


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.851     0.172      0.83      0.851     0.84     0
  0.828     0.149      0.85      0.828     0.839    1
```

```
=== Confusion Matrix ===

    a    b   <-- classified as
 6700 1169 |    a = 0
 1377 6630 |    b = 1
```