PRISCILA VRIESMAN ARAUJO

Classificação Automática de Processos em Sistemas Operacionais

Dissertação submetida ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção do título de Mestre em Informática.

Área de concentração: *Ciência da Computação*

Orientador: Prof. Dr. Carlos Alberto Maziero
Co-orientador: Prof. Dr. Júlio César Nievola

Curitiba
2011

Resumo

Em sistemas operacionais há uma grande diversidade de aplicações, programas auxiliares (associados à construção do ambiente do usuário) e serviços que executam em segundo plano, como serviços de rede e *daemons* diversos. Cada processo tem demandas próprias de processamento e de tempo de resposta, que nem sempre podem ser facilmente informadas pelo usuário ou inferidas pelo escalonador. Desta forma, o provimento de informações adicionais sobre o comportamento dos processos ao escalonador pode auxiliar significativamente sua tarefa, de forma a obter um melhor desempenho e fornecer um tempo de resposta adequado às aplicações interativas. O trabalho proposto visa explorar técnicas de mineração de dados utilizando métodos de agrupamento, seleção de atributos e classificação, aplicados à massa de informações mantidas pelo núcleo do sistema para cada processo, visando descobrir automaticamente grupos de processos com comportamento similar e classificar automaticamente novos processos nesses grupos.

Palavras-chave: classificação automática de processos, mineração de dados, escalonamento de processos.

Abstract

In operating systems there is a variety of applications, auxiliary programs (associated with the construction of the user environment) and services that run in the background, such as network services and daemons. Each process has its own demands for processing and response time, wich are not always easily informed by user or inferred by the scheduler. The supply of information about process behaviour to the scheduler can significantly help its task, yielding better system performance and adequate response time for interactive applications. Therefore, this work proposes a study to explore data mining techniques using clustering, attribute selection and classification algorithms, applied to the mass of information kept by the system kernel about each process, to automatically identify process groups with similiar behavior and to automatically classify new processes into these groups.

Keywords: process classification, data mining, scheduling.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ARFF	Attribute-Relation File Format
CFS	Completely Fair Scheduler
CFS	Correlation Feature Selection
CPU	Central Processing Unit
FIFO	First-In First-Out
GA	Genetics Algorithms
HTTP	HyperText Transfer Protocol
KDD	Knowledge Discovery in Databases
MFQ	Multilevel Feedback Queues
MLP	Multilayer Perceptron
OSFMK	Open Software Fundation MkLinux
RAM	Random-access Memory
RNA	Artificial Neural Networks
SJF	Shortest Job First
SSH	Secure Shell
VNC	Virtual Network Computing
WEKA	Waikato Environment for Knowledge Analysis

Capítulo 1

Introdução

Com a ampla disponibilidade do poder da computação nos dias atuais, através da diversidade de aplicações e necessidades dos usuários em realizar diversas tarefas (processamento de texto, navegação na Internet, execução de conteúdos de multimídias, jogos, entre outros), os sistemas operacionais devem ser capazes de realizar o controle dessas tarefas a fim de fornecer um ambiente interativo ao usuário.

Desde a criação dos sistemas de tempo compartilhado, onde se tinha um ou mais processos, o sistema operacional já tinha que prover um tempo de resposta rápido sem que prejudicasse sua produtividade. O controle de gerenciamento das tarefas em um sistema operacional é realizado pelo escalonador de processos, que verifica os novos processos solicitados pelo usuário ou processos internos do sistema e define a ordem de execução para cada processo, sem que um possa prejudicar a execução do outro. Contudo, nem sempre o escalonador pode fornecer um controle justo na execução dos processos, podendo dar prioridades a processos que estão ocupando maior tempo de processamento ou memória e não propiciando assim um tempo de resposta esperado ao usuário.

Considerando essa abordagem, a partir da década de 1980, diversas pesquisas vêm sido desenvolvidas na tentativa de reduzir custos de gerenciamento de tarefas, de forma que novas implementações algoritmos de escalonamento substituam ou melhorem os escalonadores atuais dos mais variados sistemas operacionais.

Estudos em mineração de dados, parte do processo KDD *(Knowledge Discovery in Databases)* - em português, Descoberta do Conhecimento em Base de Dados - caracterizam-se de um processo não trivial para gerar conhecimento potencialmente úteis [Fayyad et al., 1996], ou seja, consistem em buscar e interpretar, a partir de dados, padrões úteis através da aplicação de algoritmos e análise de resultados.

Desta forma, o presente trabalho propõe um estudo de classificação automática de processos utilizando técnicas de mineração de dados, com o objetivo de prover informações do comportamento dos processos para a criação de um modelo de classificação, a fim de auxiliar o

escalonador no que diz respeito a um controle justo e eficiente das prioridades de execução dos processos.

1.1 Motivação

Observando os modelos de escalonamento de processos dos sistemas operacionais atuais, nota-se que o Linux foi caracterizado por uma transição a partir de 2003, com a versão do *kernel 2.6*. Os modelos antigos de escalonamento não eram preemptivos, e tinham uma limitação significativa quando muitas tarefas estavam ativas, o que gerava uma alta carga computacional. Além disso, não havia escalabilidade, o processador poderia estar consumindo muito tempo para o escalonamento, e dedicando pouco tempo para a execução de suas tarefas.

Com a introdução *kernel 2.6*, essa política de escalonamento foi modificada. O escalonador passou a se preocupar com o tempo de processamento, interatividade com o usuário, definição de prioridades entre as tarefas e eficiência em ambiente multiprocessado.

Contudo, nem todas as características da implementação do *kernel 2.6* ou superiores, se tornam ótimas em cenários reais de aplicativos multimídia e aplicações *I/O bound*. O que faz que muitos escalonadores tenham grandes progressos no que diz respeito a maximizar a utilização da CPU, melhorar a interatividade com o usuário e fornecer um melhor suporte para arquiteturas de multiprocessadores.

Este trabalho propõe a implementação de um modelo para a classificação de processos que deve refletir o comportamento dos processos em relação às necessidades do escalonador. O modelo pode servir como mecanismo para auxiliar o estimador de interatividade do escalonador, com o foco em melhorar o tempo de resposta às tarefas interativas.

1.2 Objetivos

Este trabalho visa explorar as informações de processos através de técnicas de mineração de dados, buscando contemplar dois objetivos: a) identificar quais seriam os grupos relevantes de processos em um sistema operacional, sob a ótica do escalonador de processos, e b) classificar automaticamente os processos nesses grupos, de modo a prover informações que possam auxiliar o escalonador de processos do sistema.

Como objetivos específicos são considerados:

1. Explorar as informações disponíveis dos processos do sistema operacional Linux para construção de um modelo de classificação;

2. Identificar e definir as características mais importantes dos processos na otimização do modelo de classificação;

3. Quantificar e identificar os grupos de processos mais relevantes no contexto do escalonamento de processos através da análise do comportamento dos processos;

4. Identificar os algoritmos de classificação automática mais adequados ao contexto do trabalho proposto, tanto em relação a qualidade dos resultados quanto ao custo de processamento.

1.3 Limitações

Considerando o escopo proposto foram estabelecidos os seguintes limites para elaboração deste trabalho:

(a) não compreende a implementação de um escalonador de processos que use o conhecimento extraído pelos algoritmos de classificação, apenas indica as possibilidades de implementação ou de modificação de algum algoritmo de escalonamento;

(b) a aplicabilidade do modelo não é alcançada diretamente a todos os sistemas operacionais, pois mais estudos teriam que ser realizados nesse sentido. O estudo envolve o sistema operacional Linux, considerado o principal sistema operacional, que serve como base para outros sistemas. Cada sistema operacional possui uma arquitetura diferente, que envolveria a aplicabilidade do estudo.

1.4 Estrutura do documento

O presente trabalho apresenta-se em seis capítulos, a seguir descritos:

- O Capítulo 1 apresenta a justificativa para o desenvolvimento do modelo de classificação de processos. Neste capítulo são apresentados também os objetivos, a estrutura do documento e as limitações deste trabalho.

- O Capítulo 2 apresenta os conceitos técnicos sobre gerenciamento de processos para a compreensão do trabalho proposto.

- O Capítulo 3 expõe os conceitos e definições sobre processos do sistema operacional Linux.

- O Capítulo 4 mostra a revisão da literatura sobre mineração de dados, especialmente para métodos de seleção de características, agrupamento e classificação.

- O Capítulo 5 apresenta a metodologia para o desenvolvimento da pesquisa.

- O Capítulo 6 apresenta as considerações finais e as recomendações aos trabalhos futuros.

Capítulo 2

Gerenciamento de Processos

Este capítulo apresenta os principais conceitos sobre gerenciamento de processos, com ênfase em escalonamento de processos e seus algoritmos.

2.1 Introdução

Segundo [Etsion, 2002], as primeiras máquinas eram simples, capazes de executar somente um programa por vez, bem diferente dos dias atuais em que há necessidade do sistema operacional possibilitar a execução de vários programas simultaneamente, como processadores de texto, navegador de internet, multimídia, jogos, entre outros. Cada programa é representado por processos, e o sistema operacional deve gerenciá-los, fazendo o controle para que as tarefas sejam executadas de maneira eficiente.

O escalonador de processos é o responsável pelo gerenciamento das tarefas, controlado por algoritmos de escalonamento onde são definidos por um conjunto de regras para identificar a ordem dos processos a serem executados pelo núcleo do sistema operacional.

Nesse intuito, diversos algoritmos já foram implementados, e muitos ainda vêm sido aprimorados ou novos são criados, todos com o objetivo de se obter um melhor desempenho e tempo de resposta do sistema operacional.

2.2 Escalonamento de processos

Como os sistemas operacionais podem fornecer serviços a múltiplos usuários, ou até mesmo diversos serviços a um único usuário, é necessário que exista uma política de escalonamento para determinar as regras de como, quando e qual processo será executado na sua vez, com a finalidade de satisfazer alguns objetivos conflitantes, conforme [Tanenbaum, 2001]:

- garantir que cada processo receba uma parte justa da CPU;

- minimizar o tempo de resposta para os usuários interativos;
- minimizar o tempo que os usuários de batch devem esperar pela saída;
- maximizar o número de *batch* processados por hora;
- conciliar processos de alta prioridade com baixa prioridade.

Para tentar satisfazer esses objetivos, geralmente os sistemas operacionais baseiam-se em *time-sharing*, onde o tempo de resposta é dividido em intervalos de tempo *(quantum)* para o processo de acordo com seu algoritmo de escalonamento. Se durante a execução de um processo o *quantum* é esgotado, um novo processo é selecionado para execução, provocando então uma troca de contexto.

O escalonador Linux, como todos os outros escalonadores UNIX, privilegia os processos *I/O bound* em relação aos processos de *CPU bound* de forma a oferecer um melhor tempo de resposta às aplicações interativas. Tendo em vista que, tradicionalmente no Linux, era o *kernel* que decidia qual processo iria executar através de um método de busca linear numa lista global de processos.

Já com o Linux 2.6 [Torrey et al., 2007], a política de escalonamento foi redesenhada, com objetivo de melhorar a interatividade com a capacidade de resposta das tarefas sensíveis à latência. Ao mesmo tempo em que o escalonador realiza o escalonamento de processos, ele também estima a interatividade, que busca determinar quais são as tarefas interativas e quais são as tarefas *CPU bound*. Além disso, o escalonador Linux possibilita o ajuste dinâmico das prioridades dos processos, permitindo a mudança do comportamento do processo durante a sua execução.

2.2.1 Níveis de escalonamento

Num sistema operacional o escalonador pode ser dividido em níveis que são determinados dependendo da complexidade e da frequência das operações envolvidas. Os níveis podem ser descritos como [Stallings, 2004]:

Escalonamento de alto nível (longo prazo): determina quais tarefas deverão competir os recursos do sistema, quando selecionadas são transformadas em processos. As rotinas de alto nível são disponibilizadas pelas APIs do sistema operacional.

Escalonamento de nível intermediário (médio prazo): seleciona quais processos competirão pelo uso do processador entre os processos existentes. Este nível utiliza-se de primitivas de suspensão *(suspend)* e ativação *(activate)* para administrar a carga do sistema, correspondendo a rotinas internas do sistema operacional.

Escalonamento de baixo nível (curto prazo): determina qual processo será o próximo a utilizar o processador entre os processos ativos e em memória. É uma rotina escrita em linguagem de máquina que se encontra na memória principal, executada pelo *dispatcher* (responsável por acionar o contexto do novo processo salvando as informações do processo anterior).

Na concepção de níveis de escalonamento, o escalonamento de alto nível ocorre menos freqüentemente enquanto o escalonamento de baixo nível ocorre constantemente, dada a troca de contexto e o chaveamento do processador entre os processos ativos. A representação dos níveis de escalonamento é demonstrada na Figura 2.1

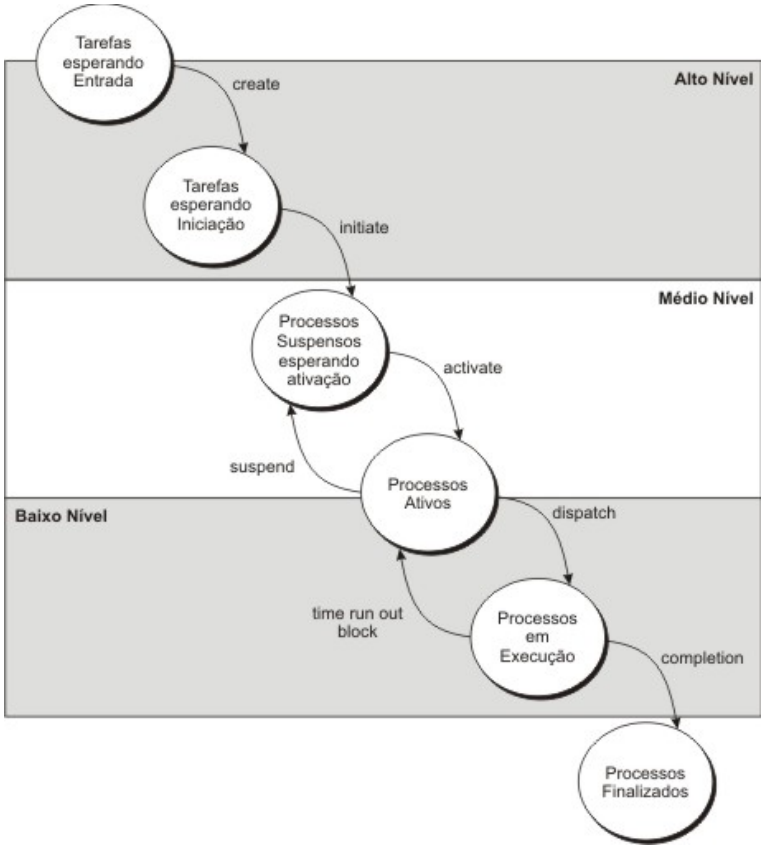

Figura 2.1: Níveis de escalonamento [Stallings, 2004]

2.2.2 Tipos de escalonamento

Num escalonamento de processos existem maneiras para tomadas de decisões que são descritas conforme os seguintes passos:

1. Quando o processo passa do estado de execução para o estado bloqueado;

2. Quando o processo passa do estado de execução para o estado pronto;
3. Quando o processo passa do estado bloqueado para o estado pronto;
4. Quando o processo termina.

Quando um novo processo está no estado pronto deve ser selecionado para utilizar a CPU, onde nos casos 1 e 4 ocorre a troca de processos na CPU obrigatoriamente. Nesse caso o tipo de escalonamento é considerado não preemptivo, caso contrário, ele é conhecido como preemptivo. De acordo com [Tanenbaum, 2001], as definições dos tipos de escalonamentos são as seguintes:

Escalonamento não-preemptivo: o processo permanece no processador, e só é liberado quando estiver terminado, retornando à fila de tarefas prontas. Esse tipo de escalonamento é adequado para processos não interativos, especialmente para *batchs*, considerado mais eficiente e previsível quanto ao tempo de entrega de suas tarefas, em que exige uma cooperação entre as tarefas para que todas possam ser executadas, sendo conhecido também como escalonamento cooperativo.

Escalonamento preemptivo: é representado pela troca de processos em execução, de forma que o processo em execução seja interrompido, ou seja, retirado do processador que será designado a um outro processo, ocorrendo assim a troca de contexto. Como a troca de contexto exige a armazenação do estado do processo, para sua posterior recuperação, pode ser apresentada uma sobrecarga computacional. Em sistemas de multiusuários interativos ou sistemas de tempo real, os processos precisam ser executados rapidamente, no qual requerem uma atenção especial, já que a natureza dos processos são imprevisíveis.

2.3 Algoritmos de escalonamento

Existem vários algoritmos que são utilizados para a realização do escalonamento de baixo nível. Em todos eles, o principal objetivo é designar o processador para um certo processo dentre vários processos existentes, otimizando um ou mais aspectos do comportamento geral do sistema.

Para realizar uma análise comparativa, foram selecionados alguns algoritmos de escalonamento dentre os principais algoritmos existentes na literatura. Além disso, esses algoritmos diferem-se em relação ao tipo de escalonamento (preemptivo, não-preemtivo), forma de implementação e aplicação nos sistemas operacionais atuais.

2.3.1 Escalonamento não-preemptivo

Escalonamento FIFO

No escalonamento FIFO *(Fist-In-First-Out)*, os processos são colocados numa fila por ordem de chegada. Cada processo em sua vez recebe o uso da CPU e permanece em execução até que seja finalizado, de forma que os demais processos na fila fiquem esperando por sua vez de processamento, como ilustrado na Figura 2.2.

Segundo [Tanenbaum, 2001], é possível ocorrer que os processos importantes fiquem em espera devido à execução de outros processos menos importantes, pois esse tipo de escalonamento não concebe qualquer mecanismo de distinção entre processos, como níveis de prioridade.

Assim, os processos de pequena duração não são favorecidos, pois seu tempo de resposta é influenciado por outros processos processados anteriormente, ou seja, o tempo de resposta aumenta consideravelmente em função da quantidade e duração dos processos na frente.

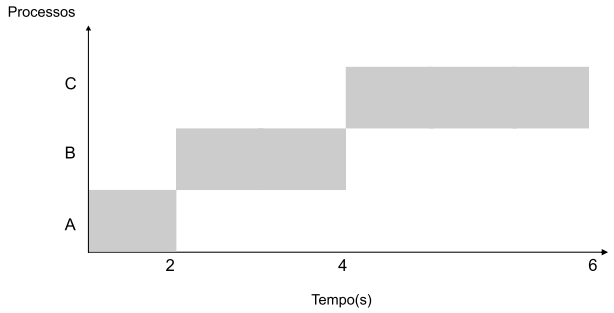

Figura 2.2: Escalonamento FIFO [Tanenbaum, 2001]

Escalonamento SJF

O escalonamento SJF *(Shortest Job First)*, funciona por um conceito simples: os processos menores terão prioridade, ou seja, serão executados primeiro, como mostrado na Figura 2.3, tendo como resultado um tempo médio mínimo de espera para cada conjunto de processos a serem executados. O cálculo de cada tempo médio é feito a partir da próxima alocação de CPU, assim o processo que utilizar a CPU por menos tempo será executado primeiro.

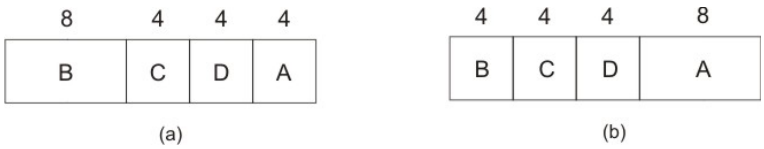

Figura 2.3: Escalonamento SJF [Tanenbaum, 2001]

2.3.2 Escalonamento preemptivo

Escalonamento Round Robin

O escalonamento *Round Robin* ou circular é um dos mais simples algoritmos de escalonamento de processos, onde os processos são organizados em fila por ordem de chegada e então enviados para execução [Tanenbaum, 2001].

No entanto, ao invés de serem executados até o fim, a cada processo é concedido apenas um intervalo de tempo *(quantum)*. Assim, caso o processo não seja finalizado neste intervalo de tempo, ocorre sua substituição pelo próximo processo da fila, e o processo interrompido é colocado no fim da fila. Nesse intervalo de tempo ocorre a preempção do processador, ou seja, o processador é designado a outro processo, sendo salvo o contexto do processo interrompido para permitir a continuidade da execução quando chegar sua vez novamente. O escalonamento *Round Robin* é mostrado na Figura 2.4:

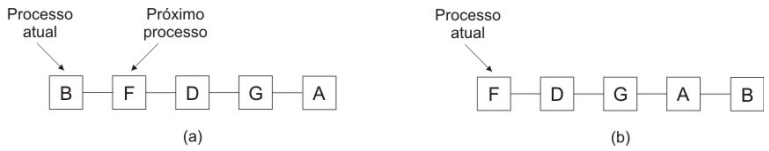

Figura 2.4: Escalonamento Round Robin [Tanenbaum, 2001]

Escalonamento por prioridade

Para esse tipo de escalonamento cada processo recebe uma prioridade, e o processo que tiver maior prioridade será executado, de acordo com [Tanenbaum, 2001]. Porém, o escalonador diminui a prioridade do processo em execução a cada interrupção a fim de evitar que os processos com alta prioridade executem indefinidamente. Se ocorrer que a prioridade do processo seja menor que a do próximo processo, é feita uma comutação de processos.

Desta forma, alternativamente pode ser atribuído um *quantum* máximo a cada processo, permitindo o uso da CPU continuamente. Quando esse *quantum* acaba, é dada uma chance para executar o próximo processo com maior prioridade. As prioridades podem ser atribuídas aos processos estaticamente ou dinamicamente. Muitas vezes é necessário que se agrupem as classes de prioridades de processos, como mostra a Figura 2.5, para que se obtenha um melhor desempenho do escalonador.

Escalonamento MFQ

O escalonamento MFQ *(Multilevel Feedback Queues)* ou filas multinível realimentadas é baseado em divisão do trabalho do processos em várias filas encadeadas, como mostra a

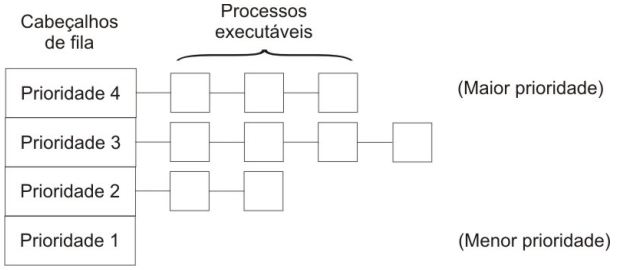

Figura 2.5: Escalonamento por prioridade [Tanenbaum, 2001]

Figura 2.6. Cada novo processo criado é colocado na fila 1, que implementa o algoritmo FIFO, onde cada processo por ordem de chegada espera por sua vez para execução [Dhotre, 2008].

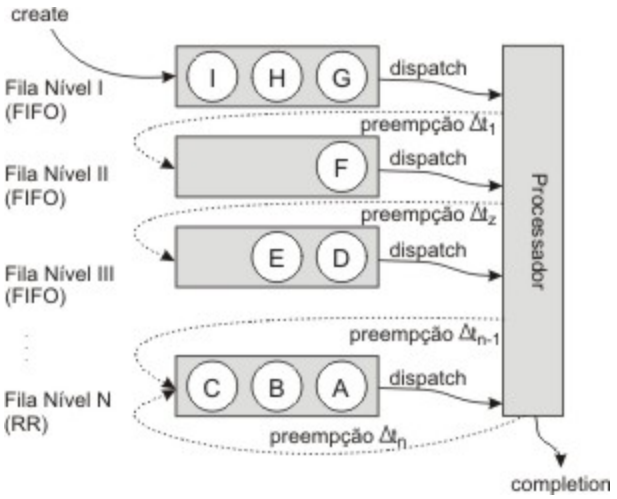

Figura 2.6: Escalonamento MFQ [Dhotre, 2008]

As filas múltiplas realimentadas se comportam como níveis de prioridade, onde as filas de nível superior são equivalentes a níveis de prioridade mais altas, e a última fila é implementada pelo algoritmo *Round Robin*, onde os processos permanecem até que sejam terminados.

A abordagem do escalonamento MFQ pode ser considerada um algoritmo adapatativo, pois é mais sensível ao comportamento dos processos, separando-os em vários níveis de fila ou em categorias, garantindo assim uma melhor eficiência.

Escalonamento CFS

O escalonamento CFS *(Completely Fair Scheduler)* consiste em tornar o escalonamento justo, de forma que quando um processo espera pela CPU, calcula-se o tempo do processo que ele ocuparia no processador, ou seja, o tempo para a execução do processo.

Segundo [Kumar, 2008], o CFS não aplica prioridade aos processos e nem identifica as tarefas interativas, ele apenas verifica o número de processos em execução na CPU, e calcula um tempo para cada processo, atribuindo um tempo muito curto para o processo, como se o processo nunca estivesse no estado bloqueado ou dormindo. O cálculo de tempo para cada processo é realizado pelo algoritmo *red-black*, uma árvore de busca binária.

O CFS implementa três políticas de escalonamento [Molnar, 2007]:

- `SCHED_NORMAL`: utilizado para tarefas regulares;
- `SCHED_BATCH`: permite que as tarefas que exigem alto tempo de processamento executem, mas que façam o melhor uso de *caches*, considerando a interatividade. É adequado para processos de tipo *batch*;
- `SCHED_IDLE`: utilizada para tarefas de baixa prioridade, onde somente é executada em ciclos em que a CPU estiver ociosa, evitando que o sistema entre em *deadlock*.

2.3.3 Comparação dos algoritmos de escalonamento

Considerando as principais características de cada algoritmo de escalonamento apresentados neste trabalho, é demonstrada uma comparação conforme a Tabela 2.1. Essas características servem como critérios para avaliação da qualidade dos algoritmos de escalonamento, que são descritas conforme:

- tipo: indica se o escalonamento é preemptivo ou não-preemptivo;
- função seleção: função de seleção utilizada para determinar qual processo na fila será selecionado para a próxima execução;
- tempo resposta: medida de tempo entre a submissão de um pedido e o início de resposta;
- produtividade: indica o nível do número máximo de tarefas processados por unidade de tempo;
- sobrecarga: indica a eficiência computacional, onde os recursos não devem ser disperdiçados.

Tabela 2.1: Comparação dos algoritmos de escalonamento

Característica	FIFO	Round Robin	SJF	Por prioridade	CFS	MFQ
Tipo	ñ preemptivo	preemptivo	ñ preemptivo	preemptivo	preemptivo	preemptivo
Função seleção	min(t chegada)	constante	min(t serviço)	complexa	complexa	complexa
Tempo resposta	alta	baixo/médio	baixo/médio	baixo	baixo	médio
Produtividade	média	baixa/média	alta	alta	alta	média
Sobrecarga	baixa	baixa	alta	média/alta	média/alta	média/alta

À medida que o desempenho está relacionado ao sistema operacional, os algoritmos de escalonamento vêm sido aprimorados, como os algoritmos mais simples: FIFO e *Round Robin*; o algoritmo FIFO não oferece garantias quanto à atrasos, podendo um processo monopolizar a CPU. Já o algoritmo *Round Robin* pode afetar os processos interativos, não sendo eficaz para sistemas de tempo compartilhado, assim como o algoritmo SJF considerado mais adequado para processos *batch*, pois o tempo de execução das tarefas precisa ser previamente conhecido.

No entanto, nos algoritmos preemptivos nota-se uma preocupação no que diz respeito ao tempo de resposta. O algoritmo Por Prioridade apresenta bons critérios, porém não garante que os processos de baixa prioridade tenham acesso a CPU. Desta forma, os algoritmos CFS e MFQ procuram a solução para os problemas dos demais algoritmos, buscando o que diz respeito à qualidade de serviço: uso de processador, tempo de resposta, tempo de permanência e produtividade.

2.4 Escalonamento de processos nos sistemas operacionais atuais

Observa-se que as versões recentes do Windows baseadas no Windows NT (incluindo XP, 2000, Vista, Server, Seven), como também para as últimas versões do MAC OS 9 e X, o escalonamento de processos é por meio do algoritmo MFQ, utilizado no Linux nas versões anteriores do kernel 2.6.

Para a plataforma Windows atual o escalonamento de processos é definido por 32 níveis de prioridade os quais vão de 0 a 31. O intervalo de 0 a 15 é definido para prioridade normal, e de 16 a 31 como prioridade para processos de tempo real.

Já o escalonador de processos do MAC OS X, de acordo com [Apple, 2011], é a modificação do escalonador OSFMK *(Open Software Fundation MkLinux)*, adicionado o tratamento de interatividade e baseado numa variante do algoritmo MFQ com divisão de filas de prioridades em 4 grupos de processos, sendo definidas de acordo com suas características:

- Normal: aplicações normais do usuário;
- Sistema: aplicações do sistema com prioridade superior as *threads* normais de usuário;
- Kernel: *threads* em espaço do núcleo que necessitam executar com uma prioridade superior às *threads* de sistema;
- Tempo real: *threads* com prioridade baseada na necessidade de reservar um espaço prédefinido de ciclos de *clock*, sendo independente de outras atividades executadas no sistema.

Numa situação em que a aplicação não faça uma operação de I/O, um processo de tempo real pode ser penalizado em sua fila de prioridades, sendo inclusive migrado para o grupo de processos Normal, visando um maior desempenho e justiça entre os processos. Os grupos de fila de maior prioridade são os processos de tempo real e de menor prioridade os processos de prioridade ao usuário Normal.

Já o Linux teve uma evolução notável desde seu início, sendo várias vezes melhorado, fazendo com que seja hoje tenha um dos melhores modelos de escalonamento. Atualmente o escalonamento do Linux em sua última versão é baseado no escalonador CFS, implementado pelo algoritmo de árvore de decisão *red-black*, que consiste em calcular o tempo que cada processo ocuparia na CPU, tornando o controle justo entre as tarefas, pois segundo [Molnar, 2007]: "O CFS basicamente modela em um hardware real um processador ideal e precisamente multitarefas".

Uma característica importante que vem sendo aprimorada nos sistemas operacionais atuais é a interatividade. Ao falar de interatividade, pode-se considerar a seguinte regra: "se um usuário for interativo, os processos também deverão ser interativos". O sistema operacional deve fornecer um tempo rápido de resposta aos processos para as aplicações de interação com o usuário, atribuindo aos processos um tempo justo de utilização da CPU, com a utilização de políticas de prioridades de processos [Love, 2003].

Com o surgimento de aplicações cada vez mais interativas com o usuário, os desenvolvedores começaram a se preocupar mais com a interatividade. No Linux, por exemplo, o nível de interatividade era um ponto fraco nas versões anteriores, e só foi possível permitir ganhos em atividades interativas de multimídia e de tempo real, com o surgimento do escalonador O(1) a partir da versão do Linux 2.6.

Já o sistema operacional FreeBSD tem um grande desempenho interativo [Hassan, 2007], pois trabalha com escalonador ULE, que possui um algoritmo baseado em filas multiníveis de tempo compartilhado, separando os processos em *threads*, e definindo as classes de *threads*, possibilitando assim dar prioridade ao tipo de processo.

Em relação ao Windows, segundo [Microsoft, 2010], teve um aprimoramento em sua interatividade de escalonamento no Windows Vista, que apresentou o serviço MMCSS *(Multimedia Class Scheduler)*. Esse serviço permite que as aplicações multimídia recebam prioridade de acesso aos recursos da CPU, garantindo seu processamento de forma rápida, sem negar os recursos da CPU para aplicações de baixa prioridade, visto a demanda pela CPU por parte de outros aplicativos em execução simultânea, onde os usuários esperam que aplicativos de multimídia propiciem uma reprodução contínua.

As versões do MAC OS X possuem um tratamento de interatividade que consiste em filas de prioridade dentro de um determinado grupo. De acordo com [Apple, 2011] há várias razões para um processo mudar sua prioridade em tempo de execução, sendo que o escalonador

deve ser capaz de fazer todo esse controle baseando-se pelo comportamento dos processos no sistema.

Assim, a interatividade no escalonador de processos é vista em diversos sistemas operacionais, de forma que proporcione um tempo justo na execução dos processos, garantindo uma melhor qualidade e desempenho do sistema ao usuário.

2.5 Trabalhos correlatos

No contexto de otimizar o escalonamento de processos do sistema operacional, alguns trabalhos já foram realizados na área, como o estudo realizado por [Negi and Kishore, 2005], que faz o uso de técnicas de aprendizagem de máquina para melhorar o desempenho do escalonador de processos no Linux, com a modificação do escalonador para minimizar o tempo de processamento na execução dos processos. Para tal, foi utilizado um algoritmo de árvore de decisão, o C4.5 para problemas de caracterização de processos, mostrando sua efetividade na otimização do escalonador, tendo como taxa de acerto entre 91% a 94%, e reduzindo o tempo de processamento na execução de processos de 1,4% a 5,8%.

Um outro trabalho na área foi relatado por [Suranauwarat and Taniguchi, 2001], onde propuseram um escalonador de processos que controla o compartilhamento de recursos na CPU observando os comportamentos dos processos. Quando um processo é executado, essa metodologia verifica as *logs* dos processos, fazendo uma verificação e o escalonador faz a troca do processo de acordo com a resposta, resultando um tempo de processamento reduzido para o escalonador.

Em [Lim and Cho, 2007] é demonstrado um método de escalonamento de processos adaptativos, que busca extrair as características dos processos em tempo real, classificando os processos em classes, e utilizando a lógica *Fuzzy* para decidir a prioridade do processo. Essa abordagem comparada com os métodos de escalonamentos tradicionais possui alguns benefícios, pois o escalonamento de processos é realizado de acordo com o tipo de processo (*batch*, *daemon*, interativo), provendo um método de escalonamento adaptativo de acordo com as preferências do usuário.

Outros estudos de escalonamento de processos em tempo real são abordados por [Nieh and Lam, 1997], que consiste num algoritmo de escalonamento de processo hierárquico para sistemas de tempo real e aplicações convencionais de tempo compartilhado, onde o tempo de cada processo é ajustado e adaptado para as aplicações.

Baseando-se em comparações dos trabalhos correlatos ao proposto, o que mais se asssemelha é o trabalho de [Negi and Kishore, 2005], que também utiliza-se técnicas de seleção de atributos, incluindo o mesmo algoritmo C4.5, porém os atributos são completamente

diferentes, pois ele se baseia em informações obtidas através dos comandos `size` e `readelf` do Linux.

Em relação aos outros trabalhos citados, este estudo possui um diferencial, pois visa identificar determinados grupos de processos por técnicas de mineração de dados baseados em suas características, não somente limitados a *batch*, *daemons* ou interativos. O intuito é verificar quais grupos podem ser relevantes no conceito do estimador de interatividade do escalonador de processos.

2.6 Conclusão

Como o gerenciamento de processos é o principal responsável pela alocação e controle de processos na CPU, é de fundamental importância o conhecimento dos algoritmos de escalonamento apresentados nesse capítulo.

Dentre os três níveis de escalonamento(alto nível, nível intermediário, baixo nível), esse trabalho foca no escalonamento de baixo nível(curto prazo), que consiste em verificar os processos que estão em memória, e seguir uma regra definida pelo algoritmo de escalonamento, selecionando o próximo processo a ser executado.

Cada algoritmo de escalonamento possui suas características particulares, que nem sempre pode fornecer um controle justo das tarefas, mas que pode ser melhorado em conjunto de outros algoritmos de escalonamento, como é o caso do algoritmo FIFO utilizado no MFQ.

A análise da comparação dos algoritmos permite a identificação dos algoritmos de escalonamento que possuem melhores desempenhos, sendo que os estudos referentes às melhorias no contexto de gerenciamento de processos são discutidos nos trabalhos correlatos.

Capítulo 3

Processos no Linux

Este capítulo apresenta sobre os conceito e características dos processos na visão do sistema operacional Linux.

3.1 Introdução

O conceito de um processo pode ser definido como abstração de um programa [Tanenbaum, 2001], ou seja, é a unidade que representa um programa no qual o sistema operacional trabalha para realizar o gerenciamento das tarefas.

De acordo com [Rusling, 1999], no Linux os processos são separados cada um com sua responsabilidade, caso um processo falhe não causará danos a outro processo. Cada processo possui um identificador único correspondente ao ID do processo (PID), e suas informações são armazenadas numa estrutura do diretório `/proc` 3.1. A execução ocorre individualmente em seus próprios endereços virtuais, onde não são capazes de interagir diretamente com outros processos, exceto se for através de mecanismos do núcleo do sistema operacional.

Um processo pode ter vida curta, permanecendo pouco tempo na CPU, como por exemplo processos de comando de texto `(man, sudo)`, ou podem ter vida longa, que são aqueles processos que demoram para serem finalizados, como processos de serviços de rede `(httpd, sshd)`.

Quando um processo é dividido em tarefas para executarem concorrentemente, essa ação é chamada de *threads*. A implementação dos processos e abstrações das *threads* no Linux possuem uma abordagem única, onde todas as *threads* são vistas como processos e um processo é simplesmente uma combinação de instrução e um conjunto de dados para realizar a execução de uma tarefa [Aas, 2005].

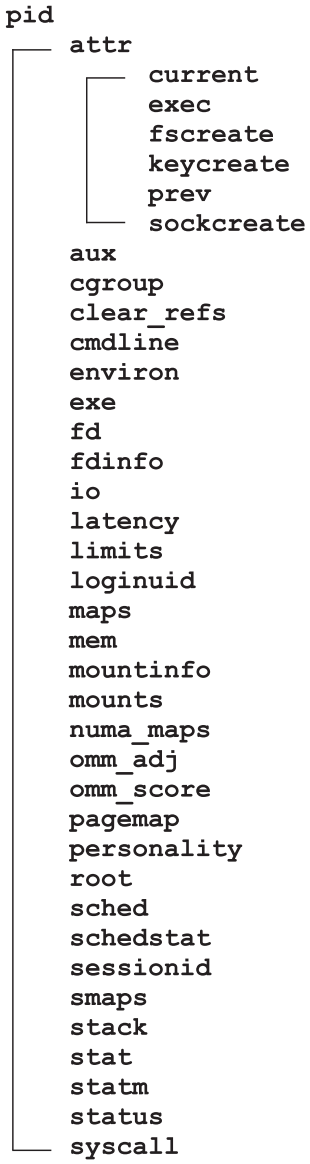

Figura 3.1: Árvore da estrutura /proc[pid]

3.2 Descritor de processos

No Linux, um processo é representado por um registro, chamado de descritor de processos, que é definido no arquivo `sched.h`, onde se encontram todas as informações sobre cada processo [Negi and Kishore, 2005]. Pode-se citar, por exemplo, que a quantidade de tempo de CPU que o processador recebe antes da execução de outro processo é armazenada no descritor de processos. Desta forma, quando todos os processos da fila de execução esgotam seu tempo de CPU, o escalonador recalcula um intervalo de tempo da CPU para cada processo. Dentre as informações armazenadas no descritor de processos, estão as seguintes:

- prioridade do processo no sistema, usada para definir a ordem na quais os processos recebem o processador;
- localização e tamanho da memória principal ocupada pelo processo;
- identificação dos arquivos abertos no momento;
- informações de tempo de processador gasto, espaço de memória ocupado;
- estados dos processos;
- contexto de execução quando o processo perde o processador, ou seja, conteúdo dos registradores do processador quando o processo é suspenso temporariamente;
- apontadores para encadeamento dos blocos descritores de processo.

3.3 Estados dos processos

Quando um novo processo é criado, não significa que ele será executado no mesmo instante, pois os processos possuem um ciclo de vida e podem estar em constante mudança, podendo migrar para vários estados durante a execução do programa. Porém os estados dos processos são mutuamente exclusivos, ou seja, todo o processo pode estar apenas em um estado num determinado momento.

Para exemplificar a mudança de estado do processo, supõe-se que um determinado processo está sendo executado, e o processador precisa executar um processo mais prioritário, assim o processo que está executando é temporariamente suspenso, e depois da execução do processo prioritário ele retorna à execução. Contudo, não são somente esses estados que os processos podem abranger, a Figura 3.2 demonstra os estados dos processos no Linux, conforme [Bovet and Cesati, 2005]:

3.4 Contexto dos processos

O sistema operacional deve armazenar as informações suficientes sobre o estado de cada processo, pois a cada intervalo de tempo, o processo é removido do processador, e ao ser executado novamente, deve retomar à sua operação na mesma posição em que parou [Tanenbaum, 2001]. Essa informação armazenada com os dados dos processos no sistema operacional é conhecida como contexto.

O contexto de um processo é composto pelos dados de espaço de endereço, pilha, endereço virtual, registro de imagem fixa, ponteiro de pilha (SP), registro de instrução (IR),

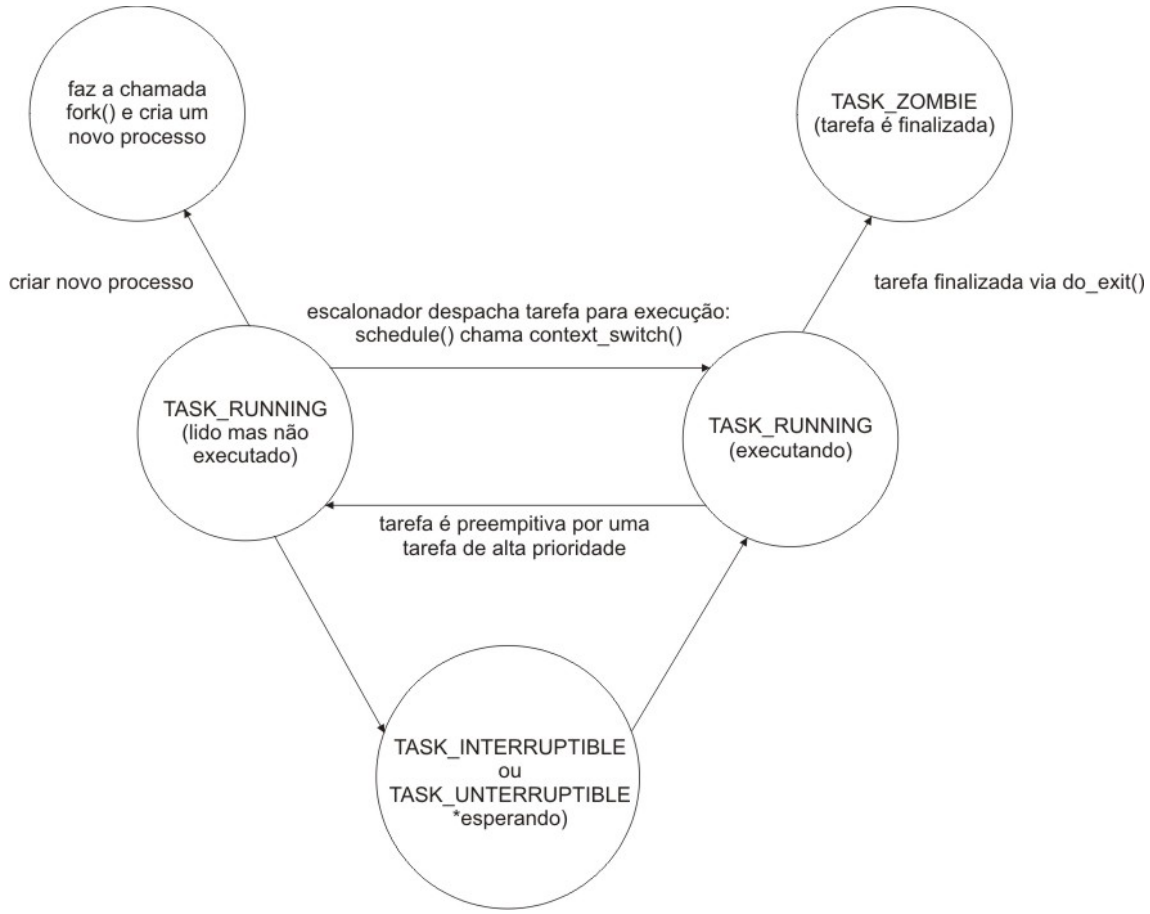

Figura 3.2: Diagrama de estado dos processos [Bovet and Cesati, 2005]

palavra de estado de programa (PSW) e outros registradores gerais do processador, de forma que sempre esteja atualizando o seu estado atual do processo.

Quando uma interrupção de processo é feita, o escalonador salva o contexto do processo na situação atual e substitui o ponteiro de tarefas atuais por um ponteiro para o contexto do novo processo definido para execução, restabelecendo seu acesso à memória e registrador de contexto.

3.5 Hierarquia de processos

Cada processo do Linux é referenciado por um processo pai, ou seja, um processo criador de outro(s) processo(s), onde ao inicializar o sistema operacional existe um processo principal que irá sempre ser executado.

Na inicialização do Linux a função `start_kernel` é ativada, sendo responsável por criar um *thread* que será o primeiro processo executado, de número 0. Na sequência, é feita a solicitação do *init* que faz a chamada de sistema `execve`, sendo o processo 1. Assim, o

sistema invoca os *initscripts*, e os processos foram uma árvore com raiz no init que tem o pid = 1.

As informações de relacionamento em árvore entre os processos ficam armazenadas nos campos `parent` e `children` no `task_struct`. Essas informações e outras informações dos processos podem ser obtidas no diretório virtual `/proc`, que armazena todas as informações dos processos ao serem iniciados.

Existem também alguns processos que são criados pelo núcleo do sistema operacional e destruídos quando o sistema é finalizado, sendo que outros podem ser carregados quando solicitados sob demanda.

3.6 Classificação de processos

Um problema que os escalonadores tem que lidar é que cada processo é único e imprevisível. Alguns processos são considerados como *I/O bound*, pois gastam muito mais tempo esperando operações de entrada e saída, e outros são os *CPU bound* que necessitam de longo tempo na CPU. Tanto os processos *I/O bound* ou *CPU bound* podem fazer parte também de diferentes classes de processos no que diz respeito às suas necessidades de escalonamento. Uma classificação usual dos processos nesse sentido consiste em separá-los em três classes:

Processos batch: Nos processos *batch*, um programa ao executar possui um conjunto de dados de entrada, realiza todo o processamento de dados e produz um conjunto de dados de saída. Esse tipo de processo é considerado de baixa prioridade em relação aos processos interativos, e freqüentemente são penalizados pelo escalonador.

Os processos *batch* não necessitam de nenhuma intervenção do usuário, e muitas vezes podem estar executando em segundo plano. Alguns exemplos de aplicações originalmente processadas em *batch* são programas envolvendo cálculos numéricos, compilações, ordenações, backups.

Processos interativos: Como os processos interativos são considerados as tarefas orientadas a entrada/ saída que interagem diretamente com o usuário, geralmente através de uma interface gráfica, precisam de um tempo de resposta rápido do sistema operacional para o tratamento de requisições no contexto do usuário.

Essa interação entre o usuário e um aplicativo pode ser também indireta através de um terceiro processo [Etsion, 2002], como por exemplo, nos sistemas UNIX, em que o teclado do usuário e entradas de mouse não são entregues diretamente à aplicação, mas sim para o servidor e o gerenciador de janelas, que são responsáveis por entregar o evento de entrada para a aplicação.

Assim, os processos interativos estão em constante interação, e quando uma ação for recebida, o sistema precisa responder de forma ágil, suspendendo qualquer processo que estiver rodando e colocar o processo interativo na CPU como prioridade, pois se o sistema demorar a responder, o usuário pode pensar que ele não está respondendo.

Processo daemons: Os *daemons* são processos do servidor inicializados no boot do sistema, que executam continuamente enquanto o sistema estiver ativo, e esperam em segundo plano até que algum serviço seja requerido, ou seja, não são projetados de forma a utilizar informações contínuas do usuário.

Para exemplificar os processos daemons, pode ser considerado um servidor HTTP passa o tempo todo respondendo requisições da rede, e normalmente não necessita intervenção do usuário, ou programas que transportam mensagens de correio de um local para outro, ou seja, os processos *daemons* são processos que tem a característica de estarem sempre executando, mas que se bloqueiam rapidamente.

3.6.1 Comparação de classes de processos

Atualmente os aplicativos estão cada vez mais interativos com o usuário, exigindo também um maior consumo de recursos (por exemplo, navegadores, tocadores de multimídia, interface gráfica), e precisam de um tempo de resposta rápido do sistema operacional. Porém o sistema operacional não conhece a origem das aplicações que possuem interação com o usuário, devido a instabilidade do comportamento da tarefa durante o ciclo de vida computacional, podendo ocorrer momentos em que o processo dependa mais da CPU.

Assim, vários processos de um sistema operacional têm características de processamento diferentes. Por essa razão, se torna de fundamental importância classificar os processos em função de suas demandas e tempo de resposta, para que seja possível aplicar as políticas de escalonamento mais adequadas a cada classe de processo.

A Tabela 3.1 descreve as principais características dessas classes.

Tabela 3.1: Características das classes de processos

Características	**Batch**	**Daemon**	**Interativo**
Interação com o usuário			•
Execução em segundo plano	•	•	
Alto custo computacional	•		
Baixo tempo de resposta			•

3.7 Conclusão

Em um sistema operacional, entender e interpretar a ações dos processos é de fundamental importância para saber o que está sendo executado, além de ser um mecanismo de estudo importante para aplicações em gerenciamento de processos.

A estrutura dos processos é diferenciada para cada sistema operacional, sendo responsável por manter todas as informações de todos os processos. No Linux a estrutura se caracteriza pelo diretório `/proc` e todo o processo possui um identificador (PID) e um processo pai (processo referente à sua criação), que possibilita um mecanismo de controle do gerenciamento das mais diversas tarefas.

Capítulo 4

Mineração de Dados e Descoberta do Conhecimento

Este capítulo apresenta sobre os conceitos do processo KDD, e particularmente sobre a mineração de dados, uma técnica confiável capaz de capturar informações escondidas em grandes volumes de dados que pode ser vista como de fundamental importância na aplicação da prática da descoberta de conhecimento em diversas áreas.

4.1 Introdução

Em geral, os sistemas computacionais armazenam uma grande quantidade de informações, e muitas vezes algumas informações úteis podem estar escondidas. No intuito de explorar dados e descobrir padrões com o objetivo de transformá-los em conhecimento útil, surgiu o KDD *(Knowledge Discovery in Databases)*, um processo de descoberta de conhecimento em bases de dados.

O processo KDD envolve diversas áreas como estatística, reconhecimento de padrões, inteligência artificial e visualização de dados, em que várias tarefas podem ser identificadas dependendo do domínio da aplicação. Entre as principais tarefas do KDD está a mineração de dados, que compreende um conjunto de ferramentas e técnicas com a finalidade de evidenciar padrões, explorando e extraindo informações num conjunto de dados.

4.2 Etapas do processo KDD

Para identificação de padrões compreensíveis o processo de descoberta do conhecimento é constituído em etapas, que envolvem todo o ciclo que o dado pode percorrer até se transformar em conhecimento. As etapas do processo KDD são apresentadas na Figura 4.1 [Fayyad et al., 1996]:

- seleção dos dados: define um conjunto de dados pertencentes a um domínio, contendo todas as possíveis características e observações (registros);
- pré-processamento e limpeza dos dados: elimina ruídos no conjunto de dados extraído, removendo dados redundantes ou inconsistentes, como também recupera dados incompletos e avalia possíveis dados discrepantes *(outliers)*;
- transformação dos dados: seleciona as características úteis para representar os dados, podendo reduzir a dimensionalidade do conjunto de dados, e realiza a formatação adequada dos dados a serem utilizados pelo algoritmo de aprendizagem;
- mineração de dados: analisa de forma automática ou semi-automática grandes bases de dados com objetivo de descobrir padrões, incluindo regras de classificação, regressão, agrupamento, modelagem de seqüência, e regras de associação;
- interpretação e avaliação: analisa os padrões descobertos a fim de que o conhecimento adquirido através da tarefa de mineração de dados seja interpretado e avaliado.

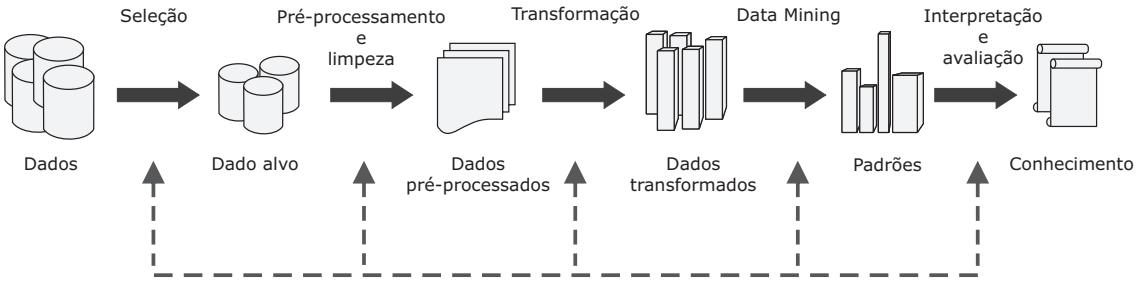

Figura 4.1: Etapas do processo KDD [Fayyad et al., 1996]

4.3 Mineração de dados

Segundo [Han and Kamber, 2000], a mineração de dados consiste num processo de descoberta de conhecimento com a utilização de técnicas e métodos que possibilitam realizar a análise de grandes quantidades no conjunto de dados para a extração de uma informação previamente desconhecida.

O processo de aprendizagem da mineração de dados funciona através de regras baseadas em observações de estados que consistem em:

- aprendizagem supervisionada: fornece exemplos ao processo de aprendizagem, sendo que cada exemplo recebe uma classe associada que constitui no modelo que caracteriza uma classe.

- aprendizagem não-supervisionada: baseia-se em observações e descobertas sem classe definida, e observa os exemplos a fim de reconhecer padrões.

Para determinar um resultado de qualidade no processo de mineração de dados, é necessário identificar o tipo de tarefa a ser aplicado na descoberta do conhecimento, de modo que seja utilizado o algoritmo corretamente atendendo à requisição da tarefa. As tarefas de mineração de dados podem ser classificadas como [Chen et al., 1996]:

- regras de associação: representado por $X \rightarrow Y$, onde X e Y são conjuntos de atributos que estão diretamente associados uns aos outros;
- padrões seqüenciais: representado por $I_1 \ldots I_n$, onde I_i é um conjunto de atributos que são apresentados em ordem cronológica de acordo com os fatos representados pelo conjunto;
- classificação ou predição: processo de encontrar um conjunto de modelos que descrevem e distinguem classes, com o propósito de utilizar o modelo para predizer a classe de objetos que ainda não foi classificado;
- análise de agrupamento *(clustering)*: identifica agrupamentos entre os dados, onde cada agrupamento é identificado por uma classe, sem ter o conhecimento prévio do modelo;
- análise de *outliers*: identifica as exceções, ou seja, os dados que não apresentam o comportamento igual da maioria.

4.3.1 Agrupamento

A análise de agrupamentos visa classificar a informação a ser utilizada pelo algoritmo de mineração de dados em grupos ou classes, baseando na similaridade das características dos dados, de forma que os dados mais similares fiquem num grupo diferente dos dados de outros grupos.

Os grupos podem representar a existência de padrões que não são naturalmente percebidos, ou seja, todo conhecimento e informações existentes que permanecem escondidas nos dados armazenados, mas que podem ser descobertos pela aplicação dos métodos de agrupamento de dados [Jain et al., 1999].

Os métodos de análise de agrupamento são categorizados como aprendizado não-supervisionado, pois se baseiam apenas nos atributos dos itens a serem classificados, não considerando os exemplos de treinamento, que possuem a finalidade de separar um conjunto de dados utilizando apenas informações dos atributos.

Convencionalmente, um agrupamento de um conjunto de itens pode ser definido como $D = x_1, x_2, \ldots, x_n$ produz uma divisão em $C = C_1, C_2, \ldots, C_k$, baseado numa certa medida de similaridade, de forma que:

- $C_i \subseteq D$ é um grupo não vazio, para $1 \leq \text{i} \leq \text{k}$
- $C_1 \cup C_2 \ldots C_k = D$
- $C_i \cap C_j = \emptyset$, para $1 \leq \text{i} < \text{j} \leq \text{k}$

De acordo com [Han and Kamber, 2000], os métodos de análise de agrupamentos podem ser divididos nas seguintes categorias:

- métodos particionais: recebem inicialmente um conjunto de itens e número de grupos a serem criados, onde vão dividindo o conjunto em grupos cada vez mais menores, até que o número de grupos desejado seja obtido. Cada grupo deve ter pelo menos um item que só pode pertencer a um único grupo, e os itens podem ser trocados entre os grupos até que sejam obtidos agrupamentos que satisfaçam uma condição de qualidade.
- métodos hierárquicos: baseados na composição e decomposição dos grupos, que podem ser divididos em aglomeração ou divisão. A aglomeração consiste em considerar de forma iterativa cada item como um agrupamento e, de acordo com a proximidade vai juntando os agrupamentos até que sejam obtidos os grupos que satisfaçam uma condição mínima de qualidade. Já o processo de divisão, consiste em iniciar com um único grupo contendo todos os itens e subdividindoo em grupos menores até que sejam obtidos grupos que satisfaçam uma condição mínima de qualidade.

DBScan

O algoritmo DBScan consiste em determinar os grupos, utilizando a idéia de densidade, que identificam as regiões de alta densidade no espaço de dados que são separadas por regiões de baixa densidade, como mostra a Figura 4.2. Esse algoritmo foi escolhido por apresentar vantagens em relação a outros algoritmos de agrupamento, com as seguintes características [Ester et al., 1995]:

- permitem descobrir agrupamentos de formatos arbitrários. A forma dos grupos podem ser de vários tipos: não-convexa, esférica, linear.
- sensíveis a ruídos, permitem, ao final do processo, classificar objetos ruidosos;
- apresentam bom desempenho em grandes bases de dados.

O DBScan inicia arbitrariamente o ponto que não tenha sido visitado, encontrando os pontos vizinhos dentro de um espaço denominado como *raio eps*. Se o número dos vizinhos for maior ou igual ao número mínimo de pontos (*minPoints*), um novo grupo é criado. Assim, o processo se repete recursivamente para todos os pontos vizinhos, até que sejam identificados os grupos. As propriedades do algoritmo DBscan são demonstradas na Tabela 4.1:

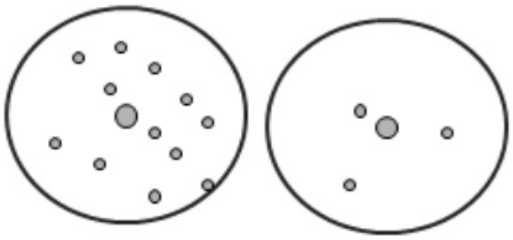

Figura 4.2: Agrupamento com DBScan

Tabela 4.1: Propriedades do algoritmo DBScan

Valor eps	Valor minPoints	Resultado
alto	alto	Poucos grupos grandes e densos
baixo	baixo	Muitos grupos pequenos e menos densos
alto	baixo	Grupos grandes e menos densos
baixo	alto	Grupos pequenos e densos

4.3.2 Classificação

A classificação consiste em utilizar dados sobre o passado (conjunto de treinamento) para classificar dados futuros (conjunto de execução de um modelo de classificação).

De acordo com [Chen et al., 1996], o processo de classificação envolve a construção de um modelo para que possam ser aplicados os dados ainda não classificados visando categorizá-los numa classe pré-definida, baseando-se nas características comuns entre o conjunto de dados da base de dados de treinamento, e gerando um rótulo para cada dado submetido na classificação.

Cada algoritmo de classificação possui uma abordagem de construção diferenciada, podendo ser através de árvores de decisão ou métodos probabilísticos, por exemplo. Entre os algoritmos de classificação existentes, foram selecionados: Redes Neurais, Naive Bayes, C4.5 e OneR, que serão também utilizados na metodologia proposta neste trabalho.

Redes Neurais

As redes neurais (RNAs) são inspiradas na estrutura do cérebro humano, com o objetivo de apresentar características similares a eles: aprendizado, associação, generalização e abstração. Sendo compostas por neurônios artificiais (processadores), altamente interconectados, que efetuam um número pequeno de operações simples e transmitem seus resultados aos processadores vizinhos.

Segundo [Alsmadi et al., 2009], a arquitetura das redes neurais é tipicamente organizada em camadas, em que cada camada tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói o padrão que será a resposta. As camadas

intermediárias funcionam como extratoras de características, seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação do problema.

As redes neurais de várias camadas são conhecidas como redes perceptron multi-camadas MLP *MultiLayer Perceptron*, e são treinadas pelo algoritmo *backpropagation*. A Figura 4.3 ilustra a estrutura das redes neurais de multi camadas.

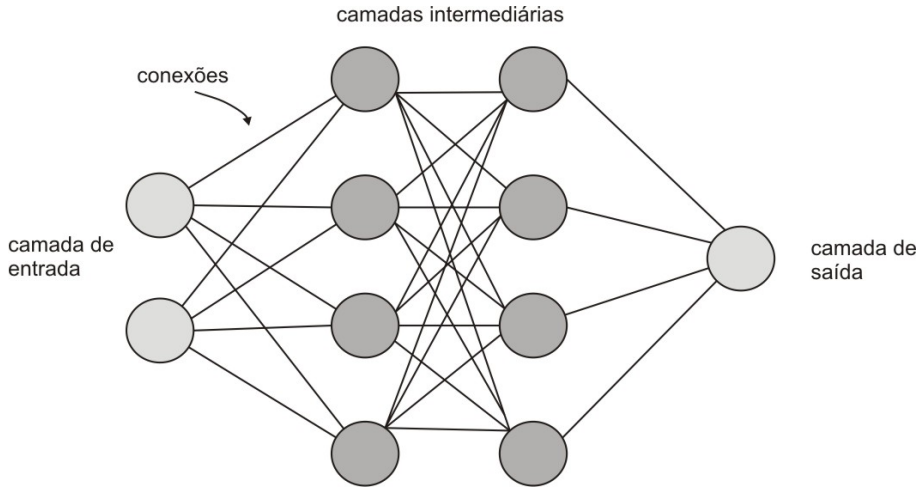

Figura 4.3: Redes neurais [Alsmadi et al., 2009]

Naive Bayes

O algoritmo Naive Bayes é um dos mais simples classificadores probabilísticos. As probabilidades são estimadas pela contagem de frequência de cada atributo do vetor de características das instâncias da base de dados de treinamento. Esse algoritmo é baseado no produto das probabilidades individuais para os valores de características da instância [Goldberg and E., 1989].

A cada instância criada, o classificador estima a probabilidade dessa instância pertencer a uma classe específica, onde para saber o cálculo exato é utilizado o teorema de Bayes, que tem como objetivo determinar a relação entre uma probabilidade condicional e sua inversa.

C4.5

O algoritmo C4.5 é baseado em árvores de decisão, em que realiza uma análise da base de dados de treinamento para a construção da árvore. O ponto de partida para a construção da árvore é a raiz da árvore, em que são realizados testes sucessivos a patir do nó da raiz, de forma que a estimativa de erro seja mais precisa se os nós filhos de um determinado nó *n* forem

eliminados, onde o nó *n* passará a ser um nó novo da folha e os nós filhos serão eliminados [Goldberg and E., 1989].

Esse algoritmo é fundamentado pela comparação de taxas de estimativas de erro de cada sub-árvore do nó da folha, onde o critério de avaliação utilizado é o ganho de informação. O pseudocódigo desse algoritmo é demonstrado conforme:

Listing 4.1: Pseudocódigo do algoritmo C4.5

```
Para cada atributo a
      Encontrar o ganho de informação da divisão a
Deixar o a_best ser o atributo com o melhor ganho de informação normalizado
Criar um nó de decisão que divide o a_best
Passar recursivamente nas sublistas obtidas através da divisão em a_best
Adicionar os nós como filhos do nó
```

OneR

O algoritmo OneR é baseado por regras de associação, que define para cada atributo da base de dados de treinamento um regra específica e seleciona a regra com menor percentual de erro, que será escolhida como uma regra única [Goldberg and E., 1989].

A escolha pela regra se define pela classe mais frequente do conjunto de valores dos atributos. Quando duas ou mais regras possuírem o mesmo percentual de erro, uma regra é escolhida por acaso. O pseudocódigo desse algoritmo é demonstrado conforme:

Listing 4.2: Pseudocódigo do algoritmo OneR

```
Para cada atributo A
      Para cada valor V desse atributo, criar uma regra:
            1. Contar quantas vezes cada classe aparece
            2. Encontrar a classe mais frenquente em c
            3. Criar uma regra "se A = V entao C = c"
Calcular o erro da hipótese baseada em A
Escolher a hipótese com menor erro
```

4.4 Seleção de atributos

Dentre as etapas do processo KDD, uma das formas de obter uma eficácia em todo o processo e uma melhor compreensibilidade do conhecimento extraído, quando se lida com espaços de alta dimensionalidade, é reduzir o número de atributos do conjunto de dados extraído.

Considera-se que muitos atributos não são utilizados na classificação, de acordo com [Jain et al., 1999], existem dois motivos para reduzir a dimensionalidade do conjunto de dados: reduzir o custo de processamento e aumentar a acurácia na classificação.

Assim, a seleção de atributos consiste em encontrar o menor subconjunto por meio de seleção dos atributos candidatos baseado numa estratégia de busca, onde cada atributo candidato do subconjunto é avaliado e comparado com o próximo melhor de acordo com o critério de avaliação [Liu and Yu, 2005]. A geração do subconjunto é um processo contínuo, sendo repetido até completar o critério de parada.

Desta forma, todo o processo de seleção de atributos consiste basicamente na nomeação dos atributos, geração do subconjunto, avaliação do subconjunto, critério de parada e validação do resultados, como mostra a Figura 4.4.

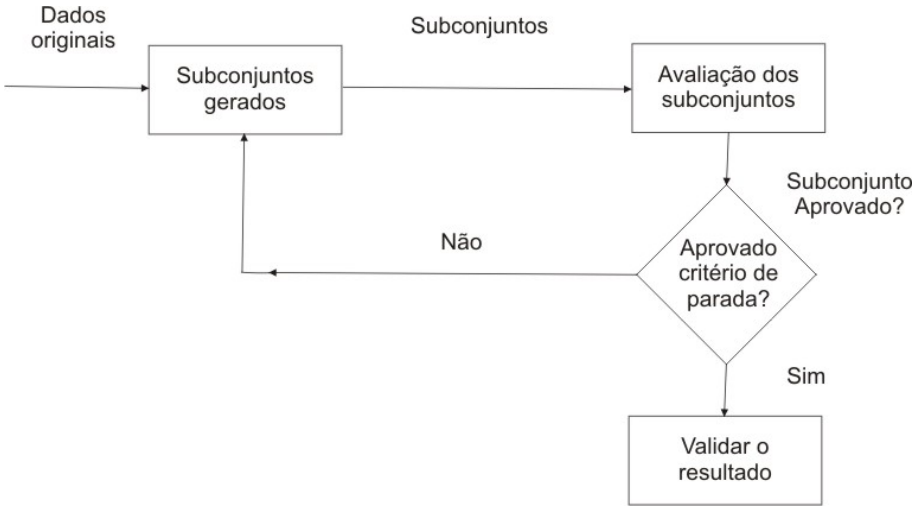

Figura 4.4: Processo de seleção de atributos [Liu and Yu, 2005]

4.4.1 Métodos de seleção de atributos

Na etapa de geração dos subconjuntos, o algoritmo de aprendizagem precisa saber se o atributo selecionado é bom, seguindo algum critério de avaliação, como: distância, dependência, precisão, consistência, entre outros. Essa abordagem para a geração dos subconjuntos pode ser realizada de duas maneiras diferentes: filtro e *wrapper*.

Filtro

O método filtro ocorre antes da fase de pré-processamento e tem como objetivo escolher um subconjunto de atributos independente do algoritmo de classificação, buscando estimar a qualidade dos atributos pelas características gerais do conjunto de dados, com a seleção de alguns atributos e exclusão de outros. O processo filtro elimina assim os atributos irrelevantes antes do aprendizado [Kohavi and John, 1997], como mostra a Figura 4.5

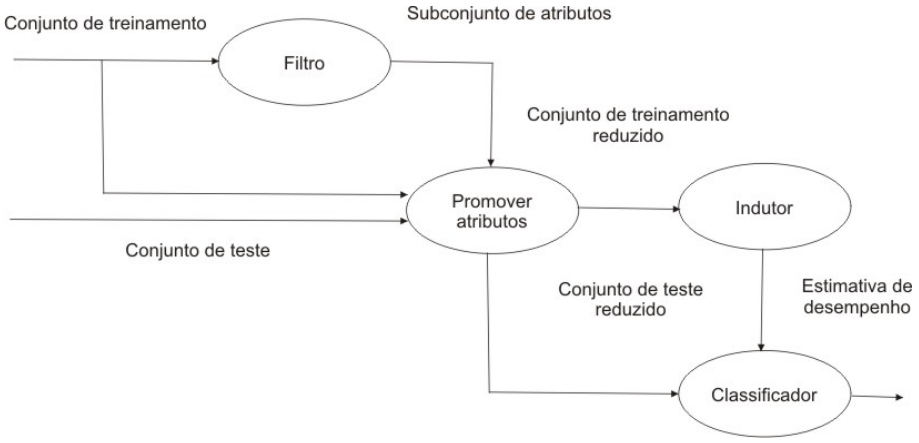

Figura 4.5: Método filtro [Freitas, 2002]

Wrapper

O método *wrapper* funciona como uma caixa preta que executa o algoritmo de aprendizagem no conjunto de treinamento e gera um subconjunto de atributos, utilizando a precisão resultado do classificador induzido para avaliar o subconjunto de atributos. Para cada subconjunto esse processo é repetido e só é finalizado quando chega no critério de parada [Kohavi and John, 1997]. O processo do método *wrapper* é demonstrado na Figura 4.6

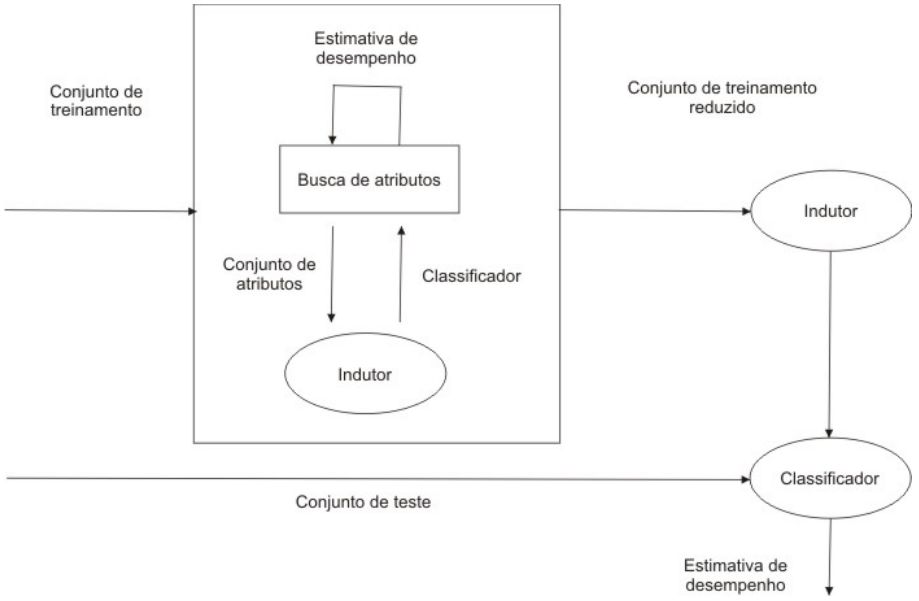

Figura 4.6: Método wrapper [Freitas, 2002]

4.4.2 Algoritmos de seleção de atributos

Os algoritmos de seleção de atributos têm como objetivo de descobrir os atributos que são relevantes para o conceito a ser aprendido. Sua terminologia pode ser descrita como [Kudo and Sklansky, 2000]:

- conjunto de atributos: um atributo inicial recebe Y, $|Y| = n$ e a seleção do subconjunto de atributos por X, onde busca o melhor subconjunto X de tamanho m. X_i denota o subconjunto de atributos de tamanho i.
- critério: a função de critério $J(X)$ avalia o melhor subconjunto X com base na capacidade do classificador discriminadas as classes de espaço de atributos representados por X. O maior valor de J indica o melhor atributo do subconjunto.

Os algoritmos de seleção de atributos selecionados englobam os dois métodos de seleção: filtro (*Ranker*, *Rank Search*) e *wrapper* (Busca Genética, *Best First*).

Filtro

Ranker: O algoritmo *Ranker* é um método de busca para avaliação individual de atributos dentro de um conjunto, não só selecionando um atributo, mas também removendo níveis hieráquicos de atributos inferiores, podendo restringir um limite que os atributos serão descartados, ou especificando quantos atributos devem ser selecionados [Witten and Frank, 2005].

Rank Search: O algoritmo *Rank Search* seleciona um atributo avaliador dentro de um subconjunto de atributos que é utilizado como base de comparação com os demais atributos do subconjunto. Caso a avaliação desse atributo seja melhor que o próximo atributo do subconjunto, o atributo é selecionado e adicionado num subconjunto de atributos selecionados [Witten and Frank, 2005].

Wrapper

Busca genética: Baseado em seleção natural e genética, o algoritmo de busca genética busca fornecer mecanismos poderosos e robustos de busca adaptativa explorando informações históricas para encontrar novos pontos de busca de melhores desempenhos. Segundo [Beasley et al., 1993], é considerado um algoritmo simples do modo de vista biológico, no qual são incluídos parâmetros de tamanho de população, número de gerações e probabilidades de cruzamento e mutação.

Best First: O *Best First* é um algoritmo de busca heurística que tem como objetivo explorar um grafo com cálculo de estimativas heurísticas de nós promissores de acordo com a regra especificada [Kohavi and John, 1997]. É iniciado por um nó inicial e mantém um conjunto de caminhos candidatos, sempre escolhendo por estimativa de cálculo o caminho candidato considerado mais curto para expansão.

4.4.3 Métricas de avaliação

Os métodos de avaliação trabalham juntamente com os algoritmos de seleção de atributos que têm como objetivo selecionar um subconjunto ou um valor númerico para orientar a busca do melhor conjunto.

Na seleção de atributos os métodos podem selecionar os atributos por meio de avaliação individual ou avaliação de subconjuntos de atributos. De acordo com [Liu and Yu, 2005], na avaliação individual os métodos são ordenados considerando sua importância na discriminação das classes, esperando que os atributos redundantes tenham a mesma importância com a eliminação dos atributos considerados irrelevantes. Já na avaliação dos subconjuntos podem ser removidos tanto os atributos irrelevantes quanto os redundantes em busca do menor subconjunto de atributos.

Nessa abordagem, existem diversos algoritmos para a avaliação dos subjconjuntos, entre eles são apresentados o *InformationGain* e CFS.

InformationGain

O algoritmo *InformationGain* é usado em árvore de decisão, baseado em entropia, que tem como objetivo avaliar os atributos de acordo com ganho de informações no que diz respeito à classe. Os valores dos atributos são divididos em partes num conjunto que realiza o cálculo da entropia, ou seja, a estimativa de aleatoriedade da variável na classe, medindo a redução da entropia [Goldberg and E., 1989].

CFS

O algoritmo CFS *(Correlation-based Feature Selection)* classifica os subconjuntos de atributos com medidas de avaliação de capacidade preditiva de cada atributo individualmente e o grau de redundância entre os atributos, selecionando o subconjunto de atributos que são diretamente correlacionados com a classe [Goldberg and E., 1989]. É composto de duas etapas: (a) avaliação da correlação entre os atributos e da correlação entre atributos e classe e (b) busca por subconjuntos de atributos e avaliação desses subconjuntos.

4.5 Conclusão

A idéia principal desse capítulo foi de apresentar de forma sucinta as principais técnicas e conceitos sobre mineração de dados a serem utilizadas na metodologia do estudo de classificação de processos proposto neste trabalho.

A mineração de dados pode ser definida como uma parte do processo KDD, que envolve o meio pelo qual os padrões são extraídos ao uso de métodos inteligentes para a aquisição de novos conhecimentos, sendo amplamente aplicados em diversas áreas que buscam encontrar respostas ou extrair conhecimentos interessantes para tomadas de decisão.

Capítulo 5

Estudo da Classificação de Processos

Este capítulo descreve os detalhes da implementação do modelo de classificação automática de processos do sistema operacional Linux por meio da aplicação do processo KDD e de métodos de mineração de dados.

5.1 Considerações iniciais

Diversos trabalhos têm demonstrado que as informações obtidas dos processos podem ser empregadas numa análise de conhecimento. Nesse propósito, as informações sobre os processos compõem todas as características e comportamentos, nos quais o conhecimento pode ser extraído.

A classificação de processos num sistema operacional pode ser considerada um problema complexo, devido à grande quantidade de processos com funções distintas e com características semelhantes que podem se tornar difíceis de serem classificados.

Desta forma, esse estudo visa explorar metodologicamente as possibilidades de aplicação de técnicas de mineração de dados à massa de informações dos processos mantidas pelo núcleo do sistema operacional Linux.

5.2 Metodologia

Em mineração de dados há diversas formas de aquisição do conhecimento com o objetivo de descobrir padrões, no qual para não induzir a resultados tedenciosos, parte-se do pressuposto que não há conhecimento prévio sobre a quantidade de grupos possíveis no processo de criação do modelo. Por isso, a metodologia proposta neste trabalho é dividida em duas etapas distintas: (a) com aprendizagem não supervisionada, que busca indícios que possam relacionar processos entre si e construir um conjunto arbitrário de grupos; e (b) com aprendizagem supervisionada, que busca classificar os processos em grupos previamente definidos.

No processo de criação do modelo, os métodos utilizados de mineração de dados foram baseados em estudos da literatura da área, e por possuírem características de construção distintas. A construção do modelo de classificação de processos é alcançada através das seguintes atividades:

1. *Extração dos atributos*: coletar informações dos processos presentes em um sistema Linux, gerando a base de treinamento;

2. *Agrupamento*: identificar os grupos existentes na massa de dados, através de um algoritmo de aprendizagem não-supervisionada;

3. *Análise de resultados*: analisar manualmente os grupos de processos obtidos na fase de agrupamento;

4. *Rotulação*: identificar a qual grupo pertence cada amostra e gerar uma base com os dados rotulados (denominada 'base completa');

5. *Seleção de atributos*: reduzir a dimensionalidade do vetor de atributos através de algoritmos de seleção de atributos, gerando novas bases de treinamento;

6. *Classificação*: utilizar algoritmos de classificação para realizar o treinamento das amostras sobre as bases reduzidas;

7. *Geração do modelo*: verificar os melhores desempenhos dos algoritmos, avaliando os resultados da classificação e definir o que melhor se enquadra em relação às taxas de acerto e tempo de processamento para o objetivo proposto.

A Figura 5.1 demonstra o método utilizado para a construção do modelo de classificação.

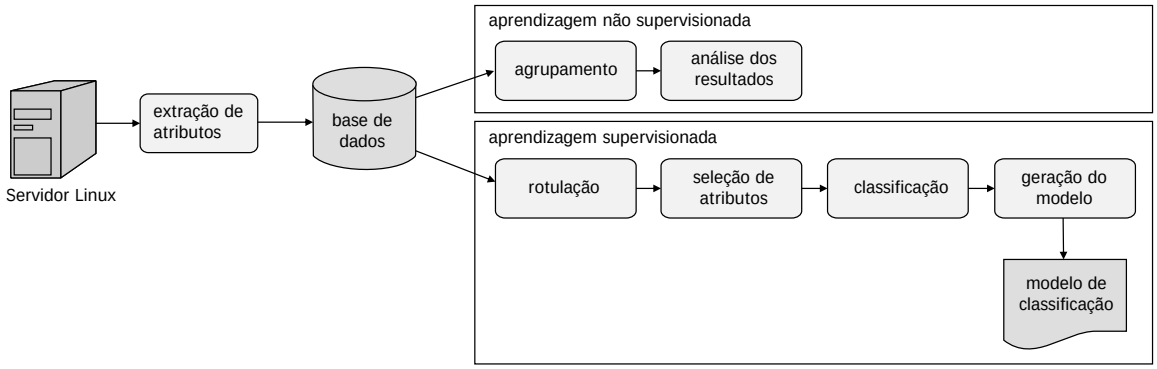

Figura 5.1: Geração do modelo de classificação

5.3 Extração de atributos

A fonte de dados utilizada para os experimentos foi um servidor de terminais remotos SSH/VNC usado pelos alunos de graduação da PUCPR para seus trabalhos acadêmicos. O servidor executa a distribuição Linux Fedora Core 12, e tem como hardware um computador Sun V20Z com dois processadores AMD Opteron 248, 4GB de memória RAM, dois discos rígidos SCSI de 72 GB cada e duas interfaces de rede *Gigabit Ethernet*. O acesso ao mesmo se dá através de sessões remotas em modo texto (SSH) e gráfico (VNC), usando ambientes de desktop gráfico Gnome, OpenBox, LXDE e XFCE. O servidor possui cerca de 800 usuários cadastrados, mas a quantidade de usuários conectados simultaneamente é muito variável, observando-se com frequência mais de 10 usuários, e esporadicamente mais de 30 usuários simultâneos. Esse ambiente foi escolhido como fonte de dados pela riqueza e diversidade de processos que ele oferece: aplicações interativas, programas de cálculo intensivo, tratamentos em *batch*, um grande número de *daemons* e outros tipos de processos.

Na extração de atributos para a construção da base de dados, foram coletados dados dos processos do diretório virtual `/proc` do servidor Linux [Faulkner and Gomes, 1991]. Nesse diretório virtual, cada subdiretório `/proc/nnn` contém arquivos com várias informações sobre o processo cujo PID é nnn. Os dados foram coletados por um programa desenvolvido em Perl, que extrai as informações dos processos, as formata em atributos e gera um arquivo em formato ARFF, conforme o Exemplo 5.1. Esse é o formato padrão de entrada da ferramenta WEKA[1][Bouckaert et al., 2002], usada nas demais etapas deste trabalho. Os dados foram coletados duas vezes por dia (às 9:00 e às 21:00) durante 185 dias, formando uma base de dados com 97.391 amostras distintas, com 108 atributos cada.

Listing 5.1: Exemplo do formato do arquivo ARFF

```
1  @relation process
2
3  @atribute stat.session numeric
4  @atribute stat.stime numeric
5  @atribute stam.data numeric
6  @atribute sched.prio numeric
7
8  @data
9  10001, 145, 5676, 44563, 0
10 10002, 157, 3445, 37754, 0
11 10003, 174, 7846, 56672, 0
12 10004, 166, 4734, 66732, 1
```

[1] O WEKA *(Waikato Environment for Knowledge Analysis)* contém ferramentas para pré-processamento, classificação, regressão, agrupamento, regras de associação e visualização de dados. Pode ser considerada uma coleção de algoritmos de aprendizagem de máquina para tarefas de mineração de dados.

5.4 Análise, agrupamento e rotulação

Com o objetivo de descobrir padrões nas amostras dos processos capturados, foram utilizadas técnicas de agrupamento a fim de possibilitar um melhor entendimento do conjunto de dados para a definição dos grupos. Cada grupo deve considerar os processos tenham características sememelhantes entre si.

Devido ao não conhecimento do comportamento dos processos, a iniciativa da escolha do algoritmo de agrupamento foi baseada no conceito de densidade, ou seja, para uma amostra com *d* atributos pode representado como um espaço *d-dimensional* e o grupos correspondem a subconjuntos de dados que estejam próximos. O algoritmo selecionado foi o DBScan, que tem como objetivo realizar agrupamentos baseado na densidade espacial, agrupando os objetos de forma arbitrária, separando os grupos de baixa densidade dos de alta densidade [Ester et al., 1996].

A principal vantagem do algoritmo DBScan é a escolha arbitrária de grupos, pois não necessita de definição de número de grupos como parâmetro. Entretanto, alguns parâmetros são configurados como medida de densidade entre os grupos para avaliação e verificação dos resultados. Os experimentos realizados são demonstrados na Tabela 5.1 com os seguintes parâmetros:

Tabela 5.1: Resultados dos testes com o algoritmo DBScan

Testes	**Valor eps**	**Valor minPoints**	**Número de grupos**
Teste 1	1.4	10	8
Teste 2	1.5	15	6
Teste 3	1.5	30	6
Teste 4	1.6	22	4

Em todos os experimentos, observou-se que ao utilizar o mesmo número de *eps* (medida de proximidade) foi possível obter o mesmo número de grupos. Assim, com a avaliação dos resultados gerados por esses experimentos foi possível identificar alguns grupos de processos, sendo observado que sempre os processos relativos a *threads* do núcleo do sistema operacional se mantém isolados, e que os demais tipos de processos, como *batch*, *daemons* e interativos, na maioria das vezes se encontravam visivelmente separados nos grupos identificados.

A análise dos resultados obtidos através do agrupamento usando DBScan permitiu identificar sete grupos distintos de processos, que receberam os seguintes rótulos em função de suas características e principais atribuições no contexto do sistema operacional:

- A (Aplicações interativas): todos os tipos de processos interativos, como os editores de texto, navegadores Web, clientes de e-mail, e outros *(a.out, chrome, codeblocks, evince, file-roller, firefox, gcalctool, geany, gedit, gnome-terminal, gnotravex, mousepad, nautilus, npviewer, pcmanfm, scalc, soffice, swritter, thunar, tomboy, wireshark, xterm).*

- D (*Daemons*): processos que executam em segundo plano e estão prontos para receber instruções *(atd, avahi-daemon, console-kit-daemon, crond, cupsd, dbus-daemon, dbus-launch, devkit-disks-daemon, devkit-power-daemon, evolution-data-daemon, failban-server, gam-server, gconfd, gnome-keyring-daemon, gnome-settings-daemon, gvfsd, hald, ibus-daemon, ibus-gconf, ibus-x, im-settings-daemon, in.talkd, init, irqbalance, lxde-settings-daemon, menu-cached, mrtg, notification-daemon, pcscd, polkitd, rsyslogd, sendmail, udevd, xfconfd, xfdesktop, xfsettingsd, xinetd).*

- F (Funcionalidades de desktop): processos que realizam tarefas de apoio ao ambiente de desktop gráfico, como processos de configuração ou framework de componentes *(bonobo-activati, canberra-gtk-pl, clock-applet, exo-helper, fih-applet, gdm-binary, gmd-session-work, gdm-simple-greeter, gdm-simple-slave, gnome-panel, gnome-screensaver, gnome-session, gweather-applet, lxpanel, lxsession, multiload-applet, notification-area, openbox, trashapplet, wnck-applet, xfce-panel, xfce-session, xfce-menu-plug, xfwm, xorg, xvnc).*

- N (*network*): processos envolvidos com a comunicação de rede *(httpd, sftp-server, snmpd, sshd).*

- C (Comandos de texto): processos de comandos simples de terminal em modo texto *(awk, bash, cb_console_runn, ck-xinit-session, consolerhelper, man, mingetty, ping, python, run-mozilla, screen,sh, ssh, ssh-agent, tail, talk, top, uml_mconsole, uml_switch, vim, wish, java, sudo).*

- K (*Kernel threads*): *threads* internas do núcleo do sistema operacional *(aio0, aio1, asyncmgr, ata0, ata1, ata_aux, bdi-default, bluetooth, cpuset, crypto0, crypto1, edac-poller, events0, events1, ext4-dio-unwrit, flush, jbd2sda1-8, jbd2sda2-8, kacpi_hotplug, kacpi_notify, kacpid, kauditd, kblockd0, kblockd1, khelper, khubd, khungtaskd, kintegrityd0, kintegrityd1, kjournald, kmpath_handlerd, kmpathd0, kmpathd1, kpsmoused, kseriod, ksmd, ksnapd, ksoftirqd0, ksoftirqd1, kstriped, ksuspend_usbd, kswapd0, kswapd1, kthreadd, mpt0, mpt_poll_0, netns, pm, scsi_eh_0, scsi_eh_1, scsi_eh_2, scsi_eh_3, sync_supers, usbhid_resumer).*

- O (Outros): processos que não se enquadram nos demais grupos *(xOrg, cupsd, Xvnc, fail, asyncmgr, linux, kblockd0, fail2ban-server).*

A Figura 5.2, mostra o gráfico de quantidade de amostras por rotulação. O conjunto das amostras obtidas demonstram diferenças entre o número de amostras para cada classe, podendo ser considerada uma base de dados desbalanceada no domínio de classificação, ou seja, quando existem muito mais casos de algumas classes do que outras.

De acordo com [Batista et al., 2004], o problema das classes desbalanceadas, surge porque os algoritmos de classificação assumem diferentes erros como igualmente importantes,

supondo que as distribuições são relativamente equilibradas, o que pode prejudicar a acurácia do modelo gerado.

Observando a aplicação real, constata-se que a quantidade de processos *deamons* é proporcionalmente mais alta do que as outras classes, o que podereia levar ao classificador perder a sua capacidade de discriminação, classificando todos os exemplos como pertencentes à classe dominante. Porém, ao considerar a quantidade de classes *versus* o número de amostras, o desbalancemento não deve afetar negativamente ao classificador.

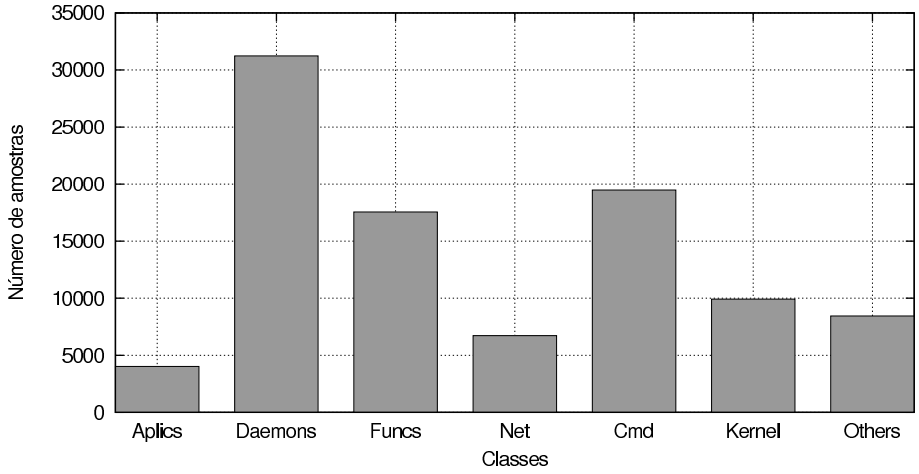

Figura 5.2: Gráfico de quantidade de amostras por rotulação

Após a etapa de definição dos grupos, a base de dados foi rotulada, ou seja, cada amostra da base foi acrescida de um atributo que representa a classe do respectivo processo (A, D, F, N, C, K , O), o que a torna preparada para a avaliação dos métodos de aprendizagem supervisionada para seleção de atributos e classificação automática.

5.5 Seleção de atributos

A seleção de atributos é uma etapa de pré-processamento que tem como objetivo descobrir quais atributos são os mais importantes, no sentido de descrever de maneira completa e não-redundante os dados da base [Liu and Yu, 2005]. No processo de descoberta dos subconjuntos de atributos foi utilizada a abordagem filtro, que busca encontrar os atributos relevantes, ou seja, que apresentam alta correlação com as classes e baixa correlação com outros atributos, já que os atributos considerados irrelevantes podem atrapalhar a classificação, além de aumentar o custo computacional e o espaço de armazenamento.

De acordo com [Kohavi and John, 1997], os algoritmos da abordagem filtro têm como objetivo selecionar um subconjunto de atributos que preserve as informações relevantes do conjunto inteiro de atributos. Os algoritmos utilizados para a busca do melhor subconjunto foram:

Busca Genética(BG), *Ranker*(RK), *Rank Search*(RS) e *Best First*(BF), e como procedimentos de avaliação foram escolhidos os métodos *Information Gain* e CFS [Liu and Yu, 2005].

Os algoritmos de busca genética são baseados em mecanismos de seleção natural e genética, que adotam uma estratégia de busca paralela e estruturada, explorando as informações históricas para encontrar novos pontos de busca. Já os métodos de busca Ranker e Rank Search são baseados em estratégia de ranking, selecionando os primeiros melhores [Bouckaert et al., 2002]. Por fim, o método de busca *Best First* é baseado em pesquisa heurística, que tenta prever o melhor atributo candidato de acordo com regras específicas. Contudo, apesar de possuírem os mesmos objetivos, esses métodos de seleção possuem implementações distintas e podem produzir subconjuntos de atributos distintos.

Assim, no intuito de facilitar o processo de classificação e avaliar o desempenho da classificação de acordo com a dimensionalidade dos atributos, foram criadas cinco novas bases de dados, formadas através da seleção de atributos. A Figura 5.3 mostra o processo de geração dos subconjuntos a partir da base de dados inicial (base completa), sobre a qual foram aplicados os métodos de seleção de atributos, gerando quatro bases de dados distintas (base BG, base RS, base RK, base BF).

Observando os resultados obtidos pelos algoritmos de seleção de atributos, foi possível identificar atributos comuns pertencentes a todas as bases de dados o que permitiu a criação de uma nova base (base AC - Atributos Comuns). Os atributos que compõem cada base de dados estão descritos a seguir:

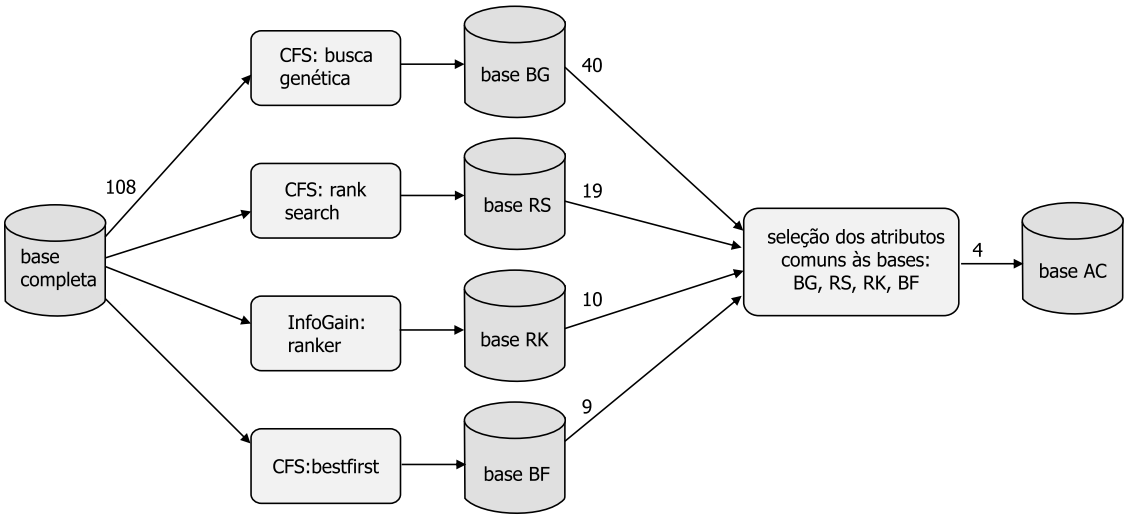

Figura 5.3: Processo de geração dos subconjuntos

Base completa : formada pelos 108 atributos coletados: *fdinfo:openfiles, io:cancelled_write_bytes, io:rchar, io:read_bytes, io:syscr, io:syscw, io:wchar,*

io:write_bytes, maps:numlines, oom_score, sched:avg_atom, sched:avg_per_cpu, sched:nr_switches, sched:nr_voluntary_switches, sched:policy, sched:prio, sched:sched_info.bkl_count, sched:se.avg_overlap, sched:se.avg_running, sched:se.avg_wakeup, sched:se.block_max, sched:se.block_start, sched:se.exec_max, sched:se.exec_start, sched:se.iowait_count, sched:se.iowait_sum, sched:se.load.weight, sched:se.nr_failed_migrations_affine, sched:se.nr_failed_migrations_hot, sched:se.nr_failed_migrations_running, sched:se.nr_forced2_migrations, sched:se.nr_forced_migrations, sched:se.nr_involuntary_switches, sched:se.nr_migrations, sched:se.nr_wakeups, sched:se.nr_wakeups_affine, sched:se.nr_wakeups_affine_attempts, sched:se.nr_wakeups_local, sched:se.nr_wakeups_migrate, sched:se.nr_wakeups_remote, sched:se.nr_wakeups_sync, sched:se.sleep_max, sched:se.sleep_start, sched:se.slice_max, sched:se.sum_exec_runtime, sched:se.vruntime, sched:se.wait_count, sched:se.wait_max, sched:se.wait_start, sched:se.wait_sum, stat:blocked, stat:cmajflt, stat:cminflt, stat:comm, stat:cstime, stat:cutime, stat:delayacct_blkio_ticks, stat:endcode, stat:exit_signal, stat:flags, stat:kstkeip, stat:kstkesp, stat:majflt, stat:minflt, stat:nice, stat:num_threads, stat:pgrp, stat:pid, stat:policy, stat:ppid, stat:priority, stat:processor, stat:rss, stat:rsslim, stat:rt_priority, stat:session, stat:sigcatch, stat:sigignore, stat:signal, stat:startcode, stat:startstack, stat:starttime, stat:state, stat:stime, stat:tpgid, stat:tty_nr, stat:utime, stat:vsize, stat:wchan, statm:data, statm:resident, statm:share, statm:size, statm:text, status:FDSize, status:nonvoluntary_ctxt_switches, status:Tgid, status:TracerPid, status:VmData, status:VmExe, status:VmHWM, status:VmLib, status:VmPeak, status:VmPTE, status:VmRSS, status:VmSize, status:VmStk, status:voluntary_ctxt_switches.

Base BG : formada por 40 atributos selecionados pelo método *Busca Genética*, com avaliação por CFS: *fdinfo:openfiles, io:rchar, oom_score, sched:avg_atom, sched:prio, sched:se.iowait_sum, sched:se.load.weight, sched:se.nr_forced2_migrations, sched:se.nr_wakeups, sched:se.nr_wakeups_sync, sched:se.sleep_start, sched:se.wait_start, stat:cstime, stat:endcode, stat:flags, stat:num_threads, stat:pgrp, stat:pid, stat:priority, stat:rss, stat:rsslim, stat:rt_priority, stat:sigcatch, stat:sigignore, stat:startcode, stat:starttime, stat:state, stat:tpgid, stat:utime, stat:vsize, stat:wchan, statm:resident, statm:share, statm:size, statm:text, status:TracerPid, status:VmHWM, status:VmLib, status:VmPeak, status:VmStk.*

Base RS : formada por 19 atributos selecionados pelo método *Rank Search*, com avaliação por CFS: *sched:se.nr_failed_migrations_affine, stat:endcode, stat:flags, stat:kstkeip, stat:sigcatch, stat:sigignore, stat:startcode, stat:state, stat:tgpid, stat:tty_nr, stat:vsize, stat:wchan, statm:size, statm:text, status:TracerPid, status:VmExe, status:VmLib, status:VmPeak, status:VmSize.*

Base RK : formada por 10 atributos selecionados pelo método *Ranker*, com avaliação por *Information Gain*: *oom_score, stat:endcode, stat:vsize, statm:size, statm:text, status:VmData, status:VmExe, status:VmLib, status:VmPeak, status:VmSize.*

Base BF : formada por 9 atributos selecionados pelo método *Best First*, com avaliação por CFS: *stat:endcode, stat:ppid, stat:sigignore, stat:startcode, stat:tty_nr, stat:vsize, stat:wchan, statm:text, status:VmLib.*

Base AC : formada pelos 4 atributos que formam a interseção dos atributos das bases BG, RS, RK e BF: *stat:vsize, stat:endcode, statm:text, status:VmLib.*

No processo da geração das bases de dados para todos os métodos de seleção de atributos foram identificados três principais grupos de atributos dos processos: *stat*, *statm*, *status*. Esses grupos contém informações de controle de diferentes estatísticas sobre o sistema, status de memória em páginas e status do sistema.

Pode-se considerar que quanto maior o número de atributos para representar uma classe, maior será o poder discriminatório do classificador. Contudo, esse fato pode contribuir para a degradação na acurácia dos resultados da classificação, que poderá ser melhorado através da aplicação do conceito da dimensionalidade: diminuir o custo de processamento e aumentar a acurácia da classificação.

Com esse objetivo, mesmo com a geração dos subconjuntos pelos algoritmos de seleção de atributos, foi gerada a base de dados AC, que consiste em observar os atributos comuns em todos os subconjuntos, ou seja, os atributos que contém informações relevantes para a separabilidade das classes. O significado de cada um desses atributos é descrito conforme:

- *stat:vsize*: memória virtual em uso pelo processo;
- *statm:text*: tamanho da memória usada pelo código do processo;
- *stat:endcode*: endereço abaixo do qual o código do processo pode executar;
- *status:VmLib*: espaço usado pelas bibliotecas compartilhadas do processo.

Em todas as bases geradas com exceção à base AC, observa-se a semelhança dos seguintes atributos demonstrados na Tabela 5.2:

Observa-se que todos os atributos em comum estão relacionados ao uso de memória pelo processo, seja o tamanho total do processo, o tamanho de seu código ou de suas bibliotecas compartilhadas, ou ainda a região da memória em que o processo pode executar. Este é um

[1]O nome de cada atributo indica sua localização; assim, o atributo *sched:se.nr_migrations* indica o valor do campo `se.nr_migrations` dentro do arquivo `/proc/PID/sched`. Contudo, alguns atributos representam valores agregados, como *fdinfo:openfiles*, que indica o número de entradas no diretório `/proc/PID/fdinfo`, ou seja, o número de descritores de arquivos abertos pelo processo. O significado de cada atributo pode ser consultado no Apêndice A - Sistema de arquivos `/proc` do Linux, e em [Hradílek et al., 2011].

Tabela 5.2: Atributos comuns selecionados das classes de processos

Atributos	**Base AC**	**Base BF**	**Base RK**	**Base RS**	**Base BG**
stat.vsize	●	●	●	●	●
stat.endcode	●	●	●	●	●
statm.text	●	●	●	●	●
status.vmLib	●	●	●	●	●
stat.tty_nr		●		●	
stat.startcode		●		●	●
stat.sigignore		●		●	●
stat.wchan		●		●	
statm.size			●	●	●
status.vmPeak			●	●	●
status.vmExe			●	●	
oom_score			●		●
stat.tgpid				●	●
stat.flags				●	●
stat.sigcatch				●	●
stat.tracerPid				●	●

resultado interessante, pois os estimadores de interatividade dos escalonadores em uso normalmente usam métricas baseadas no comportamento temporal dos processos e em sua demanda de processador [Hussein et al., 2004, Torrey et al., 2007].

5.6 Classificação

A tarefa de classificação tem como objetivo classificar atributos comuns de bases de dados em diferentes classes, sendo um modelo capaz de predizer em que classes se enquadram os novos exemplos, a partir da análise de um conjunto de dados de treinamento [Chen et al., 1996]. Como muitos métodos de classificação têm sido propostos na área de mineração de dados, para avaliação na classificação de processos, foram utilizados algoritmos comumente usados em problemas de mineração de dados e por possuírem características de construção distintas.

Os algoritmos de classificação utilizados foram C4.5, OneR, MLP e Naive Bayes. Cada algoritmo de classificação utilizado possui sua implementação específica, e são avaliados de acordo com os resultados obtidos, levando em consideração o tempo de treinamento e o percentual de acerto. Os experimentos a serem realizados com os classificadores são descritos na seção 5.7.

5.7 Experimentos realizados

Os experimentos realizados para a metodologia proposta foram basedos no treinamento de todas as bases selecionadas descritas na seção 5.5, conforme são demonstradas na Figura 5.4, consistindo na avaliação pontual da eficácia na classificação de processos através dos algoritmos escolhidos para a classificação supervisionada.

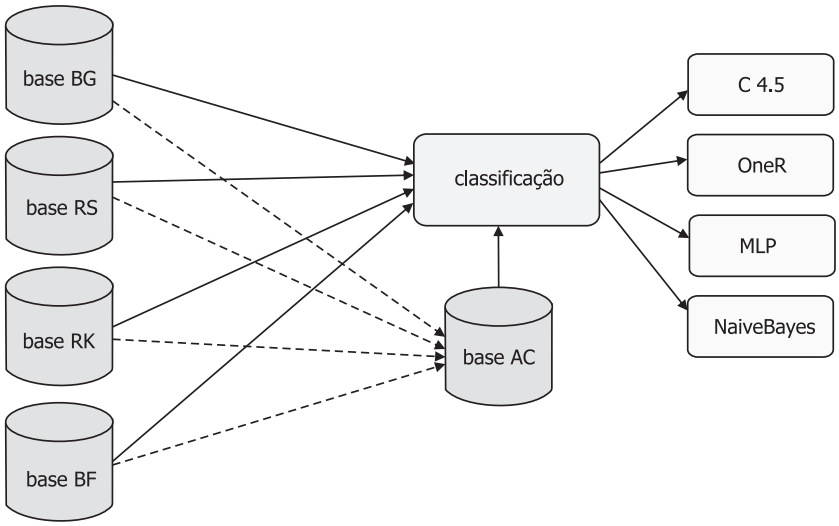

Figura 5.4: Algoritmos de classificação de processos

Para evitar a necessidade de gerar novos dados para avaliar os métodos de classificação, foi adotada a técnica de *hold-out*. Essa técnica consiste em dividir a base original em duas partes: uma parte para treinamento, ou seja, a construção do modelo, composta por aproximadamente 2/3 dos dados originais e a outra parte, com o 1/3 restante, utilizada para avaliar a qualidade do modelo gerado com os dados de treinamento.

Uma vantagem da técnica *hold-out* é que trabalha bem quando a base possui uma grande quantidade de dados, sendo uma técnica simples e computacionalmente rápida.

Cada base de dados criadas através dos métodos de seleção de atributos (base BG, base RS, base RK, base BG, base AC) foi induzida a experimentos dos classificadores: C4.5, OneR, MLP e Naive Bayes. Todos os experimentos foram realizados a partir da implementação dos classificadores na ferramenta Weka, onde foram submetidos os arquivos ARFF (formato utilizado pelo Weka), preparados com os dados extraídos e formatados para execução dos algoritmos.

Os resultados da classificação são apresentados pelas matrizes de confusão para cada classificador, que tem como objetivo gerar uma hipótese de medida efetiva do modelo de classificação, mostrando o número de classificações corretas (representadas pela diagonal principal)

versus as classificações preditas para cada classe. As matrizes de confusão podem ser consultadas no *Apêndice B - Matrizes de Confusão*.

Para o treinamento de cada classificador foram utilizados valores padrões de parâmetros fornecidos pela ferramenta Weka. A seguir são descritos os experimentos realizados para cada algoritmo de classificação:

C 4.5

Os parâmetros utilizados para o treinamento do classificador C 4.5 foram:

- confidenceFactor: fator de confiança usado para a poda da árvore. Valor = 0.25.
- minNumObj: número mínimo de instâncias por folha. Valor = 2.
- numFolds: determina a quantidade de dado utilizados para a redução de erro da poda da árvore. Valor = 3.
- seed: valor usado para randomizar o dado quando é utilizado erro de redução da poda. Valor = 1.

O classificador C 4.5 demonstrou as maiores taxas de acerto para todos experimentos realizados em relação aos outros classificadores, e um tempo de treinamento considerado baixo.

OneR

Os parâmetros utilizados para o treinamento do classificador OneR foram:

- minBucketSize: tamanho mínimo de *bucket* usado para discretizar atributos numéricos. Valor = 6.

O OneR é considerado um classificador extremamente rápido para o treinamento, e mostrou resultados estáveis para todas as bases de treinamento.

MLP

Os parâmetros utilizados para o treinamento do classificador MLP foram:

- learningRate: valor dos pesos para atualização da rede. Valor = 0,3.
- momentum: valor *momentum* aplicado aos pesos durante a atualização da rede. Valor = 0,3.
- seed: valor utilizado para inicializar um gerador de números aleatórios; Os números aleatórios são utilizados para atribuir os pesos iniciais de conexões entre os nós da rede, e também para misturar os dados de treinamento. Valor = 0.

- trainingTime: número de épocas definido ao treinamento da rede. Valor = 500.
- validationSetSize: não valida a instância da rede que será treinada por número específico de épocas. Valor = 0.
- validationThreshold: especifica o número de vezes que será feita a validação de teste da rede, até que o treinamento seja finalizado. Valor = 20.

O treinamento com o classificador MLP é considerado demorado, observou-se que quanto maior o números de atributos, maior o tempo de treinamento.

Naive Bayes

Os parâmetros utilizados para o treinamento do classificador Naive Bayes foram:

- useKernelEstimator: determina a não utilização de estimador do *kernel* para atributos númericos, e sem distrubuição normal. Valor = false.
- useSupervisedDiscretization: determina a não utilização de discretização supervisionada para converter atributos númericos em nominais. Valor = false.

O treinamento com o classificador Naive Bayes não foi tão lento quanto o MLP, mas não demonstrou boa classificação.

5.8 Resultados obtidos

Os resultados mensurados foram: 1) percentagem de acerto (número de instâncias corretamente classificadas); 2) tempo de treinamento do classificador (ou tempo de aprendizagem), levando em consideração à quantidade de atributos de cada base de dados induzida aos classificadores. As Tabelas 5.3 e 5.1 apresentam os resultados obtidos, em função ao número de atributos em cada base, mostrando que as técnicas de seleção de atributos podem influenciar positivamente nos resultados.

Constatou-se uma forte correlação entre o tempo de treinamento e o número de atributos da base utilizada pelos algoritmos de classificação, ou seja, a medida que a dimensionalidade de atributos é reduzida, o tempo de treinamento também será reduzido, pois o processamento pelo classificador será menor. Essa correlação também existe entre a taxas de acerto e o número de atributos, embora seja menos intensa e variável dependendo do classificador.

Com base nos resultados de tempo de treinamento e acurária na classificação, pode-se concluir a importância da escolha dos algoritmos, que dependendo da técnica como é o caso do Naive Bayes pode apresentar baixo desempenho, não possibilitando a eficácia de criação do modelo de classificação.

Já os algoritmos C4.5 e OneR apresentaram um ótimo desempenho, acima de 95% em todos os testes, sendo o primeiro ligeiramente melhor que o segundo. Contudo, o algoritmo OneR tem um custo computacional uma ordem de grandeza inferior ao C4.5. Ambos os algoritmos tiveram resultados satisfatórios com a base AC, que contém o menor número de atributos, e portanto um menor custo computacional.

A principal diferença entre os algoritmos OneR e C4.5 é que o primeiro fornece apenas uma única regra de classificação, utilizando um único atributo para fornecer a classificação, enquanto o segundo fornece várias regras por meio de árvore de decisão, podendo ser mais útil no sentido de apresentar as condições para a classificação em cada uma das classes encontradas.

A escolha do melhor algoritmo irá portanto depender do custo computacional que pode ser aceito, bem como da taxa de acerto exigida. Quanto ao algoritmo MLP, contata-se um percentual de acerto aceitável, porém seu tempo de processamento para a criação do modelo pode ser considerado excessivamente alto.

Tabela 5.3: Taxa de acerto dos algoritmos de classificação com *hold-out*

Bases	**Atributos**	**C4.5**	**MLP**	**OneR**	**Naive Bayes**
Base AC	4	97,31%	80,47%	95,83%	51,44%
Base BF	9	98,64%	85,81%	95,85%	51,01%
Base RK	10	97,23%	87,16%	95,85%	53,29%
Base RS	19	98,78%	83,93%	95,83%	54,97%
Base BG	40	99,86%	92,45%	96,10%	59,38%
Base completa	108	99,76%	91,32%	95,85%	58,27%

Tabela 5.4: Tempo de treinamento dos algoritmos de classificação com *hold-out*

Bases	**Atributos**	**C4.5**	**MLP**	**OneR**	**Naive Bayes**
Base AC	4	4,70s	158,88s	0,27s	0,35s
Base BF	9	8,23s	305,71s	0,58s	0,62s
Base RK	10	8,98s	317,13s	1,03s	0,84s
Base RS	19	20,42s	886,00s	1,56s	1,52s
Base BG	40	70,17s	2.280,56s	6,05s	2,93s
Base completa	108	120,24s	7.593,89s	11,48s	8,35s

5.9 Considerações finais

Os resultados obtidos neste trabalho mostram uma abordagem comparativa entre diferentes algoritmos de seleção de atributos e classificação, com o objetivo de identificar o melhor modelo para a classificação automática de processos de sistemas operacionais.

Considerada uma das mais importantes tarefas em mineração de dados, a classificação tem seu desempenho diretamente afetado por atributos que sejam redundantes ou ir-

relevantes. Observando os algoritmos utilizados: C4.5, MLP, Naive Bayes e OneR, pode-se concluir que o algoritmo C4.5 e OneR apresentam melhores resultados em relação ao percentual de acerto e tempo de treinamento; além disso, observou-se que não se torna necessário a utilização dos 108 atributos inicialmente definidos para uma boa classificação. Sendo assim, utilizando somente os 4 atributos escolhidos (`stat.vsize, statm.txt, stat.endcode, status.vmLib`) através dos métodos de seleção de atributos, pode se obter uma taxa de acerto superior a 95% com um custo computacional bastante baixo, o que é suficiente para este tipo de aplicação.

Em relação aos resultados obtidos nos experimentos, pode-se presumir que superam as expectativas. Através da grande quantidade de atributos que define o comportamento dos processos pré-definidos em classes, somente 4 dos 108 atributos coletados são suficientes para uma boa classificação. Essa característica pode servir essencialmente como motivação para abranger os estudos nessa área.

Capítulo 6

Conclusões

Em sistemas operacionais os resultados em pesquisas na área de gerenciamento de processos têm sido de grande importância para busca da otimização do escalonador de processos, no que diz respeito ao tempo de processamento, gerenciamento adequado dos recursos, desempenho do sistema e interatividade com o usuário.

O modelo de classificação de processos proposto neste trabalho busca fornecer informações úteis ao escalonador de processos do sistema operacional Linux, identificando as principais oportunidades de melhoria ao escalonamento. A estrutura geral de construção do modelo, partindo da necessidade do estudo das características dos processos e da identificação dos grupos de processos, compõe a base fundamental para a definição e análise comparativa dos diferentes algoritmos de mineração de dados utilizados.

Em termos de viabilidade de aplicação e aprimoramento de conhecimento sobre os processos, este estudo apresenta boas perpectativas, vindo do conceito do escalonador do MAC OS X que contempla a criação de quatro grupos de processos selecionados a partir de níveis de prioridade do ponto de vista do sistema operacional, muitos trabalhos recentes realizados nessa área não demonstram inovação de criação de novos grupos de processos. Esses novos grupos podem servir de forma positiva ao desempenho do escalonador de processos, como a base para criação de novas regras de priorização de processos ou mudanças no algoritmo de escalonamento do núcleo do sistema operacional.

Assim, esse trabalho propicia as seguintes contribuições:

- Estudo amplo das informações dos processos do sistema operacional Linux, identificando as características, e levando à apresentação de algoritmos de mineração de dados como solução para o problema de classificação de processos. As informações extraídas dos processos mostram-se importantes para a identificação de grupos entre os diversos tipos de processos.

- Implementação e avaliação de diferentes algoritmos de mineração de dados para a criação do modelo de classificação. Tais verificações com diferentes algoritmos se tornam importantes para o estudo na área, estabelecendo um mecanismo importante para a aquisição de conhecimento.

- Criação de mecanismos para auxiliar em termos de desempenho o escalonador de processos do sistema operacional Linux.

- Contribuição em estudos na área de sistemas operacionais, pois os resultados dos experimemtos com algoritmos de seleção de atributos indicaram que somente com os atributos relacionados à memória é possível distinguir o tipo de processo. Essa característica pode trazer benefícios à pesquisas relacionadas ao gerenciamento de processos.

- Divulgação dos resultados obtidos à comunidade científica, através da publicação do artigo [Araujo et al., 2011] no Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC).

Através da realização desse estudo, podem ser contemplados os seguintes trabalhos futuros:

- Investigar de que forma os grupos de processos definidos por esse estudo possam ser mapeados no conceito de grupos de processos definido pelo escalonador CFS *(Completely Fair Scheduler)* das versões mais recentes do núcleo Linux, como a definição de regras de prioridades dos processos, podendo se estender à alteração do `sched_fair.c` que implementa os níveis de prioridade.

- Conceituar novos grupos de processos baseados pelas classes já definidas nesse trabalho relacionadas ao assunto de adaptatividade, ou seja, evoluir o algoritmo de escalonamento de processos para regras adaptáveis ao comportamento de cada usuário, que não sejam pré-impostas os níveis de prioridade, mas que possam ser adaptáveis.

- Verificar a possibilidade de estender o estudo do modelo de classificação de processos em outros sistemas operacionais (Windows, MAC OS); além da aplicabilidade em sistemas operacionais móveis, devido ao crescimento da disponibilidade de funções multi-tarefas, características de limitação de recursos e baixo poder de processamento.

- Abranger a utilização das informações dos processos a serem aplicadas em modelos para otimização de outros recursos fornecidos pelo núcleo do Linux, como de gerenciamento de memória ou do sistema de arquivos.

- Analisar as informações do comportamento dos processos para permitir a criação de um modelo para definição de regras de alocação dinâmica de recursos para máquinas virtuais.

- Estender o estudo de mineração de dados desde trabalho, relacionados à classificação de processos para criação de mecanismos de detecção de anomalias, afim de verificar processos que possam prejudicar o desempenho ou funcionamento do sistema operacional.

- Avaliar a criação de um modelo de escalonamento de processos inteligente a partir do estudo do comportamento dos processos no contexto de utilização do sistema operacional de cada usuário. Teria que existir uma classe de "processos previsíveis", ou seja, seriam processos que entrariam em execução, antes do usuário solicitar, trazendo a interatividade ao usuário sem exigir alguma ação, considerando assim o comportamento dos processos por meio de aprendizagem, e evoluindo conforme às necessidades dos usuários.

Referências Bibliográficas

[Aas, 2005] Aas, J. (2005). Understanding the linux 2.6.8.1 CPU scheduler. *SGI*, 22:5.

[Alsmadi et al., 2009] Alsmadi, M., Omar, K., and S., N. (2009). Back propagation algorithm: The best algorithm the multi-layer perceptron algorithm. *IJCSNS International Journal of Computer Science and Network Security*, 9:378–383.

[Apple, 2011] Apple (2011). *Kernel Programming Guide - Drivers, Kernel and Hardware*. http://developer.apple.com/library/mac/documentation/Darwin/Conceptual/ KernelProgramming/KernelProgramming.pdf.

[Araujo et al., 2011] Araujo, P. V., Maziero, C. A., and Nievola, J. C. (2011). Classificação automática de processos em um sistema operacional de uso geral. *Simpósio Brasileiro de Engenharia de Sistemas Computacionais - SBESC*, 1:1–12.

[Batista et al., 2004] Batista, G. E. A. P. A., Prati, R. C., and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20–29.

[Beasley et al., 1993] Beasley, D., Bull, D. R., and Martin, R. R. (1993). An overview of genetic algorithms : Part 1, fundamentals.

[Birnbaum, 2005] Birnbaum, J. (2005). *The Linux /proc Filesystem as a Programmers' Tool*. http://www.linuxjournal.com/article/8381.

[Bouckaert et al., 2002] Bouckaert, R., Frank, E., Kirkby, R., Reutemann, P., Seewald, A., and Scuse, D. (2002). *WEKA Manual for Version 3.7.2*. University of Waikato, New Zealand.

[Bovet and Cesati, 2005] Bovet, D. and Cesati, M. (2005). *Understanding The Linux Kernel*. O'Reilly & Associates Inc.

[Chen et al., 1996] Chen, M. S., Han, J., and Yu, P. S. (1996). Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883.

[Dhotre, 2008] Dhotre, I. A. (2008). *Operating Systems*. Technical Publication Pune, 1 edition.

[Ester et al., 1996] Ester, M., Kriegel, H. P., X., J. S., and Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data mining*, volume 1996, pages 226–231. AAAI Press.

[Ester et al., 1995] Ester, M., Kriegel, H.-P., and Xu, X. (1995). A database interface for clustering in large spatial databases. In *KDD*, pages 94–99.

[Etsion, 2002] Etsion, Y. (2002). Scheduling of interactive process. Technical report, Hebrew University of Jerusalem.

[Faulkner and Gomes, 1991] Faulkner, R. and Gomes, R. (1991). The process file system and process model in UNIX System V. In *USENIX Conference Proceedings*.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34.

[Freitas, 2002] Freitas, A. A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Goldberg and E., 1989] Goldberg and E., D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[Han and Kamber, 2000] Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1st edition.

[Hassan, 2007] Hassan (2007). *ULE Scheduler in FreeBSD*. http://my.opera.com/blu3c4t/blog/show.dml/1531517.

[Hradílek et al., 2011] Hradílek, J., Silas, D., Prpič, M., Kopalová, E., Ella, Lackey, D., Ha, J., O'Brien, D., Hideo, M., and Domingo, D. (2011). *Red Hat Enterprise Linux 6 – Deployment Guide*. Red Hat Inc. Appendix C – The `proc` File System.

[Hussein et al., 2004] Hussein, N., Kolivas, C., Haron, F., and Yong, C. (2004). Extending the Linux operating system for grid computing. In *APAN Network Research Workshop*.

[Jain et al., 1999] Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323.

[Kerrisk, 2010] Kerrisk, M. (2010). *Linux Programmer's Manual (2010)*. http://www.kernel.org/doc/man-pages/online/pages/man5/proc.5.html.

[Kohavi and John, 1997] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324.

[Kudo and Sklansky, 2000] Kudo, M. and Sklansky, J. (2000). Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition*, 33:25–41.

[Kumar, 2008] Kumar, A. (2008). *Multiprocessing with the Completely Fair Scheduler - Introducing the CFS for Linux*. http://www.ibm.com/developerworks/linux/library/l-cfs/.

[Lim and Cho, 2007] Lim, S. and Cho, S.-B. (2007). Intelligent OS process scheduling using fuzzy inference with user models. In *IEA/AIE'07: Proceedings of the 20th international conference on Industrial, engineering, and other applications of applied intelligent systems*, pages 725–734, Berlin, Heidelberg. Springer-Verlag.

[Liu and Yu, 2005] Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502.

[Love, 2003] Love, R. (2003). Interactive kernel performance kernel performance in desktop and real-time applications. *Linux Symposium*, pages 285–296.

[Menage, 2004] Menage, P. (2004). *Cgroups Documentation*. http://www.mjmwired.net/kernel/Documentation/cgroups.txt.

[Microsoft, 2010] Microsoft (2010). *Multimidia Class Scheduler Service*. http://msdn.microsoft.com/en-us/library/ms684247.aspx.

[Molnar, 2007] Molnar, I. (2007). *This is the CFS scheduler*. http://people.redhat.com/mingo/cfsscheduler/sched-design-CFS.txt.

[Negi and Kishore, 2005] Negi, A. and Kishore, K. P. (2005). Applying machine learning techniques to improve Linux process scheduling. In *TENCON IEEE Region 10*, pages 1–6. IEEE.

[Nguyen, 2004] Nguyen, B. (2004). *Linux Filesystem Hierarchy: Chapter 1*. http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[Nieh and Lam, 1997] Nieh, J. and Lam, M. (1997). The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *ACM Symposium on Operating Systems Principles*, pages 506–519.

[Rusling, 1999] Rusling, D. A. (1999). *The Linux Kernel*. http://tldp.org/LDP/tlk/.

[Shalom, 2010] Shalom, H. (2010). *Creating and using proc files*. http://www.rt-embedded.com/blog/archives/creating-and-using-proc-files/.

[Smalley, 2003] Smalley, S. (2003). *Process Attribute API for Security Modules*. http://lwn.net/Articles/28222/.

[Stallings, 2004] Stallings, W. (2004). *Operating Systems: Internals and Design Principles*. Prentice Hall, 5 edition.

[Suranauwarat and Taniguchi, 2001] Suranauwarat, S. and Taniguchi, H. (2001). The design, implementation and initial evaluation of an advanced knowledge-based process scheduler. *ACM SIGOPS Operating Systems Review*, 35(4):61–81.

[Tanenbaum, 2001] Tanenbaum, A. (2001). *Modern operating systems*. Prentice Hall PTR.

[Torrey et al., 2007] Torrey, L. A., Coleman, J., and Miller, B. P. (2007). A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. *Software: Practice and Experience*, 37(4):347–364.

[Witten and Frank, 2005] Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Burlington, MA, 2st edition.

Apêndice A

Sistema de arquivos /proc do Linux

A.1 Introdução

Os sistemas operacionais contêm uma infinidade de processos que utilizam os recursos do computador para que possam realizar suas tarefas. Para isso é necessário um meio que permita controlá-los, e que armazene as informações dos processos para as mais diversas finalidades, como ferramentas de administração de sistema, geração de estatísticas, depuração de problemas, segurança e ajustes de desempenho. No núcleo do GNU/Linux essas informações estão contidas no sistema de arquivos `/proc`, que possui uma grande quantidade de informações disponíveis sobre cada processo, como o uso de memória, processador, arquivos abertos, entre outros.

O primeiro diretório `/proc` foi implementado por Tom Killian em 1984, sendo projetado para substituir a chamada ptrace (possibilita que um processo possa controlar outro processo, manipulando seus descritores de arquivo, memória, registradores). Já a versão definitiva do `/proc` surgiu em sua oitava edição em 1991, por Roger Faulkner e Ron Gomes. Nessa implementação, de acordo com [Faulkner and Gomes, 1991], os arquivos já suportavam chamadas `read()`, `write()` e `ioctl()`, com 37 `ioctl()` no total, que permitiam o controle básico de processos, como `signals/fault/syscall`, manipulação de registros e informações de status. Isso possibilitou a criação de uma poderosa base para construir ferramentas, como a `ps`, sem ter a necessidade de realizar chamadas de sistema especializadas.

Assim, a chamada de sistema `ptrace` é substituída pelo sistema de arquivos `/proc`, onde da mesma forma, um programa pode ler ou escrever dados diretamente no endereço do processo através do descritor de arquivo correspondente no `/proc`.

Segundo [Shalom, 2010], a maioria dos sistemas operacionais baseados em Unix possuem um sistema de arquivos `/proc`, que fornece uma maneira fácil de acesso às informações sobre os processos, atuando como uma interface para estrutura de dados internas do núcleo e possibilitando a alteração de parâmetros do núcleo em tempo de execução. O sistema de

arquivos `/proc` oferece muitas possibilidades de interação com as informações internas do sistema, como por exemplo:

- visualização de informações de estatísticas;
- visualização de informações de hardware;
- alteração de parâmetros em tempo de execução;
- visualização e modificação dos parâmetros de rede e host;
- visualização de informações de desempenho da memória.

A.2 Estrutura `/proc`

O sistema de arquivos `/proc` é criado toda vez que o sistema é inicializado; este contém diretórios e arquivos virtuais, onde seus conteúdos são criados dinamicamente. Um arquivo virtual pode apresentar informações do núcleo para o usuário, como também serve como um meio de enviar informações do usuário para o núcleo.

De acordo com [Birnbaum, 2005], a estrutura do diretório `/proc` é dividida em dois grupos principais de arquivos. O primeiro grupo envolve os arquivos nomeados numericamente com o correspondente ID do processo (PID) que é criado para cada processo existente. Já o segundo grupo considera os arquivos regulares, mas igualmente virtuais e dinâmicos que descrevem algumas características de operações do núcleo.

Os diretórios e arquivos virtuais referente ao primeiro grupo é representado pela A.1, e contém as seguintes informações [Kerrisk, 2010]:

- `attr`: implementado pelo módulo de segurança utilizando getprocattr e setprocattr, é composto pelos arquivos [Smalley, 2003]:
  - `current`: representa os atributos atuais do processo, pode ser utilizado para obter o contexto de segurança de um processo, como no SELinux;
  - `exec`: representa os atributos da chamada `execve`, apoiando o papel das transições, através de uma melhor segurança na inicialização do processo ou herança do estado atual do sistema.
  - `fscreate`: representa os atributos da chamadas `open()`, `mkdir()`, `symlink()`, e `mknod()`.
- `auxv`: contém o conteúdo do interpretador de informações ELF, que são passados para o processo em tempo de execução.

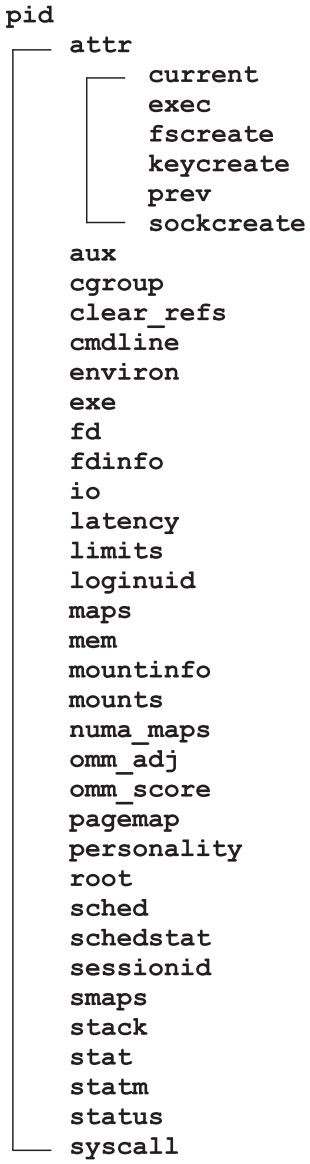

Figura A.1: Árvore da estrutura /proc[pid]

- `cgroup`: armazena as informações do mecanismo de controle para a agregação/particionamento de um conjunto de processos, e seus futuros filhos, em grupos hierárquicos [Menage, 2004].

- `cmdline`: armazena a linha de comando completa para o processo, exceto se o processo estiver no estado de *zombie*.

- `coredump_filter`: contém as informações da imagem da memória do processo no momento de sua finalização, pois a ação de certos *signals* por padrão é causar o término do processo e produzir um arquivo *dump* na memória.

- `cpuset`: informações sobre o posicionamento de cpu e memória, onde cada processo pertence a um *cpuset*. Quando um processo faz um `fork()`, o processo filho é colocado na *cpuset* de seu processo pai, como também um processo pode ser transferido de um para outro *cpuset*, o que permite que os dados de CPUs e memória existentes sejam alterados.
- `cwd`: aponta para o link simbólico do diretório de trabalho atual do processo.
- `environ`: armazena as informações do ambiente do processo.
- `exe`: a partir do Linux 2.2, esse arquivo contém o caminho real da chamada do processo executado, que aponta para o executável.
- `fd`: subdiretório contendo uma entrada para cada arquivo que o processo foi aberto, chamado pelo seu descritor de arquivo.
- `fdinfo`: subdiretório com as informações de entrada para cada arquivo que o processo foi aberto, chamado pelo seu descritor de arquivo.
- `io`: contém estatísticas de io para cada processo em execução.
- `latency`: armazena as informações de latência que são controladas pelo núcleo. Quando a latência controlada não é uma latência tradicional de interrupção, pode ocorrer que processo consuma mais recursos da CPU, e quando ocorrer uma latência de encontro do aplicativo, o núcleo pode ficar no estado *sleep* várias vezes.
- `limits`: mostra os limites dos recursos de software e hardware, e unidades de limites mensuradas para cada recurso do processo.
- `loginuid`: mantém as informações sobre os processos, incluindo o PID, UID do usuário que inicializou o processos, e seus processos filhos.
- `maps`: contém as regiões de memória mapeadas, e suas permissões de acesso.
  - `address`: endereço do mapeamento;
  - `perms`: permissões de acesso;
  - `offset`: deslocamento do arquivo mapeado;
  - `dev`: dispositivo onde está mapeada a região de memória (maior ou menor);
  - `inode`: indica se um *inode* está associado a uma região na memória, 0 (zero) indica que não está associado;
  - `pathname`: caminho do arquivo da memória mapeada.
  - `mem`: armazena as informações das páginas de memória de um processo.

- `smaps`: mostra o consumo de memória para cada um dos mapeamentos do processo:
  - `size`: tamanho do mapeamento;
  - `rss`: montante do mapeamento na RAM;
  - `shared_clean`: número de páginas públicas compartilhadas limpas;
  - `shared_dirty`: número de páginas públicas compartilhadas sujas;
  - `private_clean`: número de páginas privadas compartilhadas limpas;
  - `private_dirty`: número de páginas privadas compartilhadas sujas.
- `mountsinfo`: contém as informações dos pontos de montagem, incluindo um identificador exclusivo para cada processo, tipo, informações do sistema de arquivos, e opções de bloqueio.
- `mounts`: mostra a lista de todos os sistemas de arquivos montados no *namespace* do processo.
- `mounstats`: mantém as informações de estatísticas e configurações sobre os pontos de montagem no namespace do processo.
- `net`: contém arquivos com informações sobre parâmetros e estatística da rede do sistema, com os seguintes arquivos:
  - `arp`: tabela ARP do kernel;
  - `ATM`: contém arquivos de configurações e estatísticas *Asynchronous Transfer Mode* (ATM);
  - `dev`: lista os dispositivos de rede configurados no sistema, com as estatísticas de transmissão e recepção;
  - `dev_mcast`: exibe os diferentes grupos de *multicast*;
  - `IGMP`: lista os endereços de IP *multicast*;
  - `ip_fwchains`: armazena as informações sobre as regras, quando é utilizado o *ipchains* para filtrar pacotes;
  - `ip_fwnames`: lista de nomes da cadeia do *firewall*;
  - `ip_masquerade`: tabela de máscaras com informações do *ipchains*;
  - `ip_mrcache`: lista do roteamento *multicast* em *cache*;
  - `ip_mrvif`: lista das interfaces virtuais *multicast*;
  - `netstat`: lista as estatísticas de rede, como o tempo limite, *cookies* enviados e recebidos;

  - `psched`: lista dos parâmetros globais *scheduler*;
  - `raw`: lista as estatísticas dos dispositivos;
  - `rout`: tabela de roteamento do núcleo;
  - `rt_cache`: *cache* de roteamento atual do sistema;
  - `SNMP`: lista dos dados do protocolo *Simple Network Management Protocol*(SNMP);
  - `sockstat`: estatísticas dos *sockets*;
  - `tcp`: informações detalhadas do *socket* TPC;
  - `tr_rif`: lista do *token ring* RIF da tabela de roteamento;
  - `udp`: informações detalhadas do *socket* UDP;
  - `unix`: lista dos sockets do domínio UNIX em uso;
  - `wireless`: lista os dados da interface sem fio.

- `numamaps`: armazena informações de acesso da memória não-uniforme (NUMA), que são encontradas nos multiprocessadores cuja memória é dividida em múltiplas áreas de memória.

- `oom_adj`: ajusta a pontuação de seleção dos processos que devem ser mortos fora da memória (OOM). O valor padrão para este arquivo é 0 (zero).

- `oom_score`: mostra a pontuação atual que o núcelo dá ao processo com a finalidade de selecionar um processo para o *OOM-Killer* [1]. A pontuação mais alta significa que o processo é mais provável ser seleccionadas pelo *OOM-Killer*. O *oom_score* reflete também o ajuste mudança bits especificado pela definição *oom_adj* para o processo.

- `pagemap`: mostra a estrutura física para cada página virtual em que o processo está mapeado, com um total de 64 bits para cada página virtual:
  - `bits 0-54`: número de quadros de páginas (PFN);
  - `bits 0-4`: tipo de troca de páginas;
  - `bits 5-54`: troca de página renovada;
  - `bits 55-60`: mudança de página (se página = 1 « mudança de página)
  - `bit 61`: reservado para uso futuro;
  - `bit 62`: página trocada;
  - `bit 63`: página atual.

[1] OOM-Killer é um componente do subsistema de memória virtual do Linux, responsável por "sacrificar"processos que estejam consumindo muita memória em situações extremas de saturação da memória RAM.

- `personality`: como o Linux suporta diferentes domínios de execução, ou personalidades, esse arquivo armazena a personalidade atual que deve ser igual a 0xFFFFFFFF. Caso contrário, é armazenado o domínio de execução do processo da chamada.
- `root`: suporta a idéia de uma raiz do sistema de arquivos, definida pela chamada de sistema `chroot()`.
- `sched`: mostra as informações relacionadas ao escalonador.
  - `sleep_max`: tempo de dormência voluntariamente interrupível, que normalmente são inofensivos e não causam latências visíveis ao usuário.
  - `block_max`: tempo de dormência máximo não interruptíveis, como atividades de contenção em discos IO, ou bloqueios de *swap*.
  - `wait_max`: tempo máximo que uma tarefa tem na CPU.
- `sched_stat`: contém informações de estatística por processos individualmente.
- `sessionid`: retorna o ID da sessão que um processo é executado.
- `stack`: informações do *stacktrace* do processo.
- `statm`: informações sobre o status da memória em páginas:
  - `size`: tamanho total do programa;
  - `resident`: tamanho total do programa residente;
  - `share`: páginas compartilhadas;
  - `lib`: bibliotecas utilizadas;
  - `data`: dados e pilhas;
  - `dt`: páginas sujas.
- `status`: provê informações contidas nos diretórios *stat* e *statm* num formato mais legível.
  - `name`: nome do processo executado;
  - `state`: estado atual do processo (R: executando, S: sleeping, D: disk spleep, T: parado, Z: zombie, X: morto);
  - `tgid`: ID do grupo de *threads*;
  - `pid`: ID da *thread*;
  - `tracerPID`: PID do processo do aplicativo que está lançando o processo;

  - `uid`: UID do sistema de arquivo, real, efetivo e salvo;
  - `gid`: GID do sistema de arquivo, real, efetivo e salvo;
  - `groups`: lista de grupos complementares;
  - `FDSize`: número de *slots* alocados no descritor de arquivos no momento;
  - `vmPeak`: tamanho máximo da memória virtual;
  - `vmSize`: tamanho da memória virtual;
  - `vmLck`: tamanho da memória locada;
  - `vmHWM`: tamanho máximo da memória residente;
  - `vmRSS`: tamanho da memória residente;
  - `vmData, vmStk, vmExe`: tamanho do dados na memória;
  - `vmStk`: tamanho do *stack* na memória;
  - `vmExe`: tamanho dos segmentos de texto na memória;
  - `vmLib`: tamanho do código da biblioteca compartilhado;
  - `vmPTE`: tamanho da entrada da tabela de páginas;
  - `threads`: número de *threads* do processo;
  - `sigPnd, shdPnd`: número de *signals* pendentes para *threads* e processos;
  - `sigBlk, sigIgn, sigCgt`: máscaras que indicam *signals* de bloqueado, ignorado e capturado;
  - `apInh, capPrm, capEff`: máscara de capacidades habilitadas herdáveis, permitidas e efetivas;
  - `capBnd`: limite de capacidade;
  - `cpus_allowed`: máscara da CPU em que o processo pode ser executado;
  - `cpus_allowedlist`: máscara da CPU em que o processo pode ser executado em formato de lista;
  - `mems_allowed`: máscara dos nós de memória permitidos para o processo;
  - `mems_allowedlist`: máscara dos nós de memória permitidos para o processo em formato de lista;
  - `voluntary_context_switches, nonvoluntary_context_switches`: número do contexto de *switches* voluntários e não voluntários.

- `syscall`: verifica o número de chamada do sistema e retorna ao núcleo o processo solicitante. O valor de retorno 0 (zero) indica sucesso, e um um valor de retorno -1 indica um erro.

- `task`: diretório que contém um subdiretório para cada *thread* do processo, ou seja, para cada subdiretório contém um conjunto de arquivos com o mesmo nome e conteúdo correspondente ao processo pai.
- `wchan`: armazena o endereço da chamada de sistema, ou seja, o "canal"em que o processo está esperando.

Para o segundo grupo do sistema de arquivos `/proc`, a estrutura de arquivos possuem os seguintes arquivos e respectivas informações [Nguyen, 2004]:

- `cpuinfo`: dados sobre a CPU;
- `devices`: lista os dispositivos encontrados no sistema, como por exemplo, modem, placa de som, placa de rede, teclado, impressora, e outros;
- `cmdline`: comandos passados ao núcleo do linux na inicialização;
- `interrupts`: interrupções (IRQs) dos dispositivos;
- `ioports`: endereços das portas I/O (entrada/saída);
- `pci`: componentes PCI instalados no sistema;
- `filesystems`: sistemas de arquivos suportados pelo núcleo;
- `meminfo`: informações sobre a memória.
- `modules`: módulos carregados no núcleo;
- `mounts`: partições montadas no sistema;
- `partitions`: partições existentes e que o Linux reconheceu;
- `version`: versão do núcleo;
- `ide`: interfaces de discos ide;
- `iomem`: áreas de memória de entrada/saída;
- `kcore`: core do kernel;
- `self`: informações sobre o próprio programa;
- `sys`: arquivos que controlam o sistema;
- `uptime`: tempo que o sistema está ativo.

A.3 Conclusão

Como o diretório `/proc` disponibiliza uma grande quantidade de informações detalhadas sobre os processos, é possível obter e realizar ajustes na configuração dos processos em tempo de execução, o que proporciona uma personalização e melhor funcionamento do sistema, além de ser também um importante mecanismo de comunicação com o núcleo.

Todavia, um ponto considerado como desvantagem é a perda de portabilidade, pois cada sistema operacional Unix pode possuir uma variante própria da estrutura de sistemas de arquivos `/proc`.

Apêndice B

Matrizes de Confusão

Os resultados obtidos através dos experimentos pelos classificadores (C4.5, MLP, OneR, Naive Bayes) para cada base de dados de treinamento (AC, BF, RK, RS, BG) são demonstrados nas matrizes de confusão.

B.1 C 4.5

A tabela B.1 demonstra a matriz de confusão do algoritmo C4.5 a base de AC - atributos comuns, onde foram selecionados 4 atributos.

Tabela B.1: Matriz de confusão - Algoritmo C4.5 - Base AC

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1330	1	2	0	4	16	0
D	0	10464	1	0	0	138	1
F	0	0	6003	0	0	1	2
N	0	0	0	2318	0	0	0
C	0	0	0	0	6412	102	0
K	0	0	0	0	0	3362	0
O	5	2	7	0	6	602	2334

A tabela B.2 demonstra a matriz de confusão do algoritmo C4.5 a base de BF, onde foram selecionados 9 atributos.

Tabela B.2: Matriz de confusão - Algoritmo C4.5 - Base BF

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1346	0	2	0	4	0	1
D	0	10602	1	0	0	0	1
F	1	0	5998	0	0	0	7
N	0	0	0	2318	0	0	0
C	1	1	0	0	6507	0	5
K	0	0	0	0	0	3362	0
O	6	2	8	0	7	401	2532

A tabela B.3 demonstra a matriz de confusão do algoritmo C4.5 a base de RK, onde foram selecionados 10 atributos.

Tabela B.3: Matriz de confusão - Algoritmo C4.5 - Base RK

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1322	3	5	0	7	16	0
D	2	10463	1	0	0	138	0
F	4	2	5996	0	0	1	3
N	0	0	0	2318	0	0	0
C	4	2	0	1	6405	102	0
K	0	0	0	0	0	3362	0
O	5	3	7	1	5	602	2333

A tabela B.4 demonstra a matriz de confusão do algoritmo C4.5 a base de RS, onde foram selecionados 19 atributos.

Tabela B.4: Matriz de confusão - Algoritmo C4.5 - Base RS

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1347	0	1	0	2	0	3
D	0	10599	2	0	0	0	3
F	2	0	6001	0	0	0	3
N	0	0	0	2318	0	0	0
C	0	0	0	0	6512	0	2
K	0	0	0	0	0	3335	7
O	13	3	3	0	6	333	2598

A tabela B.5 demonstra a matriz de confusão do algoritmo C4.5 a base de BG, onde foram selecionados 40 atributos.

Tabela B.5: Matriz de confusão - Algoritmo C4.5 - Base BG

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1339	3	4	0	3	0	4
D	1	10599	0	0	1	0	3
F	1	1	6004	0	0	0	3
N	0	0	0	2318	0	0	0
C	0	2	3	0	6507	0	2
K	0	0	0	0	0	3362	0
O	5	1	7	0	7	1	2935

B.2 OneR

A tabela B.6 demonstra a matriz de confusão do algoritmo OneR a base de AC, onde foram selecionados 4 atributos.

Tabela B.6: Matriz de confusão - Algoritmo OneR - Base AC

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1269	13	15	2	20	16	18
D	19	10271	35	2	130	138	9
F	6	1	5985	0	4	1	9
N	0	0	0	2318	0	0	0
C	5	32	7	3	6362	102	3
K	0	0	0	0	0	3362	0
O	4	135	21	0	26	602	2168

A tabela B.8 demonstra a matriz de confusão do algoritmo OneR a base de BF, onde foram selecionados 9 atributos.

Tabela B.7: Matriz de confusão - Algoritmo OneR - Base BF

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1264	18	15	2	20	16	18
D	7	10283	35	2	130	138	9
F	6	1	5985	0	4	1	9
N	0	0	0	2318	0	0	0
C	5	32	7	3	6362	102	3
K	0	0	0	0	0	3362	0
O	4	135	21	0	26	602	2168

A tabela B.8 demonstra a matriz de confusão do algoritmo OneR a base de RK, onde foram selecionados 10 atributos.

Tabela B.8: Matriz de confusão - Algoritmo OneR - Base RK

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1262	18	15	2	20	16	20
D	7	10283	35	2	130	138	9
F	6	1	5985	0	4	1	12
N	0	0	0	2318	0	0	0
C	5	32	7	3	6362	102	3
K	0	0	0	0	0	3362	0
O	2	135	21	0	26	602	2170

A tabela B.9 demonstra a matriz de confusão do algoritmo OneR a base de RS, onde foram selecionados 19 atributos.

Tabela B.9: Matriz de confusão - Algoritmo OneR - Base RS

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1269	13	15	2	20	16	18
D	19	10271	35	2	130	138	9
F	6	1	5985	0	4	1	9
N	0	0	0	2318	0	0	0
C	5	32	7	3	6362	102	3
K	0	0	0	0	0	3362	0
O	4	135	21	0	26	602	2168

A tabela B.10 demonstra a matriz de confusão do algoritmo OneR a base de BG, onde foram selecionados 40 atributos.

Tabela B.10: Matriz de confusão - Algoritmo OneR - Base BG

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1264	18	15	2	20	16	18
D	7	10283	35	2	130	138	9
F	6	1	5989	0	0	1	9
N	0	0	0	2318	0	0	0
C	5	32	12	3	6355	102	5
K	0	0	0	0	0	3362	0
O	4	135	21	0	25	602	2169

B.3 MLP

A tabela B.11 demonstra a matriz de confusão do algoritmo MLP a base de AC, onde foram selecionados 4 atributos.

Tabela B.11: Matriz de confusão - Algoritmo MLP - Base AC

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	771	166	381	0	4	25	6
D	189	8741	1001	442	0	138	93
F	34	1205	4775	0	0	1	11
N	0	94	0	2224	0	0	0
C	7	743	12	5	5301	400	46
K	0	0	0	0	0	3362	0
O	108	437	195	95	23	603	1495

A tabela B.12 demonstra a matriz de confusão do algoritmo MLP a base de BF, onde foram selecionados 9 atributos.

Tabela B.12: Matriz de confusão - Algoritmo MLP - Base BF

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	886	14	424	0	38	0	11
D	10	9671	623	127	35	138	0
F	7	475	5517	0	7	0	0
N	0	226	0	2092	0	0	0
C	27	366	13	37	6070	0	1
K	0	0	0	0	0	3362	0
O	32	384	823	36	260	584	837

A tabela B.13 demonstra a matriz de confusão do algoritmo MLP a base de RK, onde foram selecionados 10 atributos.

Tabela B.13: Matriz de confusão - Algoritmo MLP - Base RK

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	854	38	426	0	4	25	6
D	0	9711	327	310	44	138	74
F	28	659	5315	0	0	1	3
N	0	109	0	2209	0	0	0
C	56	242	81	5	5727	400	3
K	0	0	0	0	0	3362	0
O	8	222	261	92	86	603	1684

A tabela B.14 demonstra a matriz de confusão do algoritmo MLP a base de RS, onde foram selecionados 19 atributos.

Tabela B.14: Matriz de confusão - Algoritmo MLP - Base RS

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	760	32	547	0	12	0	2
D	126	9694	645	0	0	138	1
F	2	1325	4678	0	1	0	0
N	0	184	0	2102	32	0	0
C	56	342	18	0	6077	0	21
K	0	0	0	0	0	3362	0
O	16	218	949	0	72	581	1120

A tabela B.15 demonstra a matriz de confusão do algoritmo MLP a base de BG, onde foram selecionados 40 atributos.

Tabela B.15: Matriz de confusão - Algoritmo MLP - Base BG

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	1169	13	156	0	10	0	5
D	16	10087	307	20	26	138	10
F	17	161	5820	0	0	0	8
N	0	54	0	2224	40	0	0
C	20	27	23	7	6399	0	38
K	0	0	0	0	0	3362	0
O	45	217	94	120	79	581	1820

B.4 Naive Bayes

A tabela B.16 demonstra a matriz de confusão do algoritmo Naive Bayes da base de AC, onde foram selecionados 4 atributos.

Tabela B.16: Matriz de confusão - Algoritmo Naive Bayes - Base AC

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	771	0	626	0	0	16	0
D	122	368	8606	1370	0	138	0
F	14	0	5991	0	0	1	0
N	0	0	94	2224	0	0	0
C	204	397	1427	9	4375	102	0
K	0	0	0	0	0	3362	0
O	846	47	1366	92	0	603	3

A tabela B.17 demonstra a matriz de confusão do algoritmo Naive Bayes da base de BF, onde foram selecionados 9 atributos.

Tabela B.17: Matriz de confusão - Algoritmo Naive Bayes - Base BF

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	725	0	625	0	0	0	3
D	123	124	8605	1614	0	138	0
F	2	0	6002	0	0	0	2
N	0	0	94	2224	0	0	0
C	725	63	749	397	4384	0	196
K	0	0	0	0	0	3360	2
O	967	0	1191	139	3	581	75

A tabela B.18 demonstra a matriz de confusão do algoritmo Naive Bayes da base de RK, onde foram selecionados 10 atributos.

Tabela B.18: Matriz de confusão - Algoritmo Naive Bayes - Base RK

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	725	0	595	0	0	16	17
D	122	240	8606	1498	0	138	0
F	90	0	5915	0	0	1	0
N	0	0	94	2224	0	0	0
C	65	397	617	9	4375	102	949
K	0	0	0	0	0	3362	0
O	71	47	1338	92	0	602	806

A tabela B.19 demonstra a matriz de confusão do algoritmo Naive Bayes da base de RS, onde foram selecionados 19 atributos.

Tabela B.19: Matriz de confusão - Algoritmo Naive Bayes - Base RS

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	730	2	601	0	18	0	2
D	123	124	8605	1499	0	138	115
F	36	0	5968	0	0	0	2
N	0	0	94	2224	0	0	0
C	209	202	582	397	5062	0	62
K	0	0	0	0	0	3360	2
O	202	0	1190	139	107	581	737

A tabela B.20 demonstra a matriz de confusão do algoritmo Naive Bayes da base de BG, onde foram selecionados 40 atributos.

Tabela B.20: Matriz de confusão - Algoritmo Naive Bayes - Base BG

	A	**D**	**F**	**N**	**C**	**K**	**O**
A	913	1	418	0	13	0	8
D	198	1859	7832	348	29	27	211
F	5	12	5986	1	0	0	2
N	0	29	94	2195	0	0	0
C	233	267	447	397	4645	0	525
K	0	0	0	0	0	3343	19
O	309	8	1052	144	114	581	748