1
Análise de desempenho de programas paralelos

Lucas Mello Schnorr - schnorr@inf.ufrgs.br[1]

Resumo:

O objetivo da análise de desempenho de aplicações paralelas é a identificação de regiões do programa que tem uma baixa exploração dos recursos computacionais. Este capítulo faz um apanhado geral das técnicas mais comuns utilizadas na análise de desempenho de programas paralelos. São vistos os conceitos básicos da área de análise de desempenho, seguido pelas técnicas de observação e de análise de desempenho. O capítulo termina através de alguns exemplos de ferramentas de coleta e análise.

[1]Lucas é doutor em computação pela Universidade Federal do Rio Grande do Sul e pelo *Institut National Polytechnique de Grenoble* (2009), através de um acordo de cotutela. Ele fez sua graduação na Universidade Federal de Santa Maria (2003). Atualmente, Lucas é professor adjunto do Departamento de Informática Aplicada da UFRGS, atuando como orientador no Programa de Pós-Graduação em Computação (PPGC). Suas áreas de interesse de pesquisa se concentram em processamento paralelo e distribuído, com enfoque na análise de sistemas computacionais e aplicações paralelas e distribuídas. Também trabalha na área de visualização da informação, sistemas computacionais, compiladores, linguagens de programação para alto desempenho, e simulação de sistemas computacionais.

1.1. Introdução

O desenvolvimento de aplicações paralelas de alto desempenho é uma tarefa complexa por diversas razões. A escalabilidade dos supercomputadores paralelos atuais, com milhares de núcleos de processamento [DON 97, DON 2013], é um dos principais fatores. A questão é como explorar todo o paralelismo presente na máquina. Outro fator que torna complexo o desenvolvimento de aplicações é o indeterminismo da execução: pelo fato que os nós de processamento são independentes, duas execuções da mesma aplicação com a mesma entrada podem ter comportamentos diferentes.

A estratégia para controlar a complexidade do desenvolvimento de aplicações paralelas envolve o conhecimento de bibliotecas de comunicação de passagem de mensagem como MPI (*Message Passing Interface*) [GRO 94], ferramentas para a exploração do paralelismo de um único nó computacional como OpenMP [DAG 98], e um bom conhecimento do sistema computacional responsável pela execução da aplicação. O desenvolvedor projeta a aplicação, prevendo a utilização dessas bibliotecas de comunicação, de uma forma que os recursos sejam explorados ao máximo para se atingir alto desempenho. O sucesso do projeto de uma aplicação paralela e do seu desempenho em tempo de execução dependem diretamente de como é feito o mapeamento entre os requerimentos da aplicação e os recursos disponíveis.

Uma vez a aplicação paralela implementada e livre de erros de programação, o desenvolvedor procede para a etapa de análise de desempenho. O objetivo da análise é verificar se a implementação obtém um bom desempenho considerando as peculiaridades e características do sistema computacional paralelo e das escolhas feitas no momento do projeto e implementação da aplicação. Esta etapa, não menos complexa que a própria implementação, se manifesta através de um processo cíclico onde o desenvolvedor realiza várias execuções experimentais da aplicação, coletando informações a respeito da execução, analisando-os e, por fim, modificando a aplicação para melhorar o seu desempenho.

A análise de uma aplicação paralela pode ser dividida em duas fases:

Fase de coleta A primeira fase, de coleta dos dados, ocorre durante a execução da aplicação e objetiva registrar informações consideradas importantes a respeito do comportamento da aplicação e do sistema. Informações que podem ser coletadas podem variar desde marcações de estados da aplicação como, por exemplo, qual função é executada e por quanto tempo, até dados do ambiente de execução como a taxa de utilização do processador.

Fase de análise dos dados A fase da análise dos dados tem por objetivo identificar problemas de desempenho durante a execução da aplicação e as suas causas. Existem várias técnicas de análise de desempenho que podem ser empregadas nesta segunda fase como, por exemplo, técnicas de visualização de rastros e análise estatística.

As duas fases do processo de análise de desempenho são feitas, na maioria das vezes, de forma separada. Primeiro coletam-se informações a respeito da aplicação a ser analisada. Depois, após o fim da execução da aplicação no ambiente paralelo, os dados coletados são analisados. Embora bastante utilizada, não há consenso nesta abordagem de separação em duas fases: trabalhos realizam a coleta e análise de forma concorrente.

Independentemente da abordagem utilizada, a análise de desempenho de uma forma geral é uma parte crucial no desenvolvimento de aplicações paralelas. Somente

através de uma rigorosa análise e com um conjunto de métricas apropriadas, o desenvolvedor será capaz de afirmar que para uma determinada configuração de supercomputador paralelo, a sua aplicação obtém alto desempenho. Caso a aplicação venha a ser portada para uma configuração de máquina diferente, o processo de análise de desempenho deve ser repetido para validar que o alto desempenho é ainda obtido nesta mudança. Na maioria das vezes nestas situações, o ciclo de análise e eventuais melhorias deve ser repetido para que a aplicação seja adaptada ao novo ambiente.

Este capítulo traz uma visão geral das técnicas mais comuns utilizadas na análise de desempenho de programas paralelos. Inicialmente são vistos conceitos básicos da área de análise de desempenho, como efeito de sonda, sincronização de relógios, análise em tempo de execução e *post-mortem*, entre outros. Em seguida são vistas as técnicas de observação e coleta de dados de monitoramento, sendo classificadas e discutidas de acordo com a complexidade e sua aplicabilidade. Por fim são vistas as técnicas de análise de desempenho feitas a partir dos dados coletados, seguida por alguns exemplos de ferramentas de coleta e análise.

Presença na Escola Regional de Alto Desempenho

Um minicurso sobre depuração de programas paralelos foi apresentado na primeira edição da Escola Regional de Alto Desempenho (ERAD), no ano de 2001, pelo professor Benhur de Oliveira Stein (UFSM). Desde então, as técnicas de análise de desempenho evoluíram substancialmente e estas mudanças estão refletidas neste texto principalmente nas Seções 1.4. e 1.5.. As técnicas de observação e coleta de comportamento, assim como os conceitos básicos da área não mudaram significativamente mas são relembrados aqui sob uma nova organização. O texto apresentado aqui é complementar àquele apresentado em 2001 [STE 2001].

Estrutura do capítulo

Este capítulo está organizado da seguinte forma. A seção 1.2. apresenta os conceitos básicos da área de análise de desempenho. A Seção 1.3. apresenta as principais técnicas de observação e registro do comportamento de aplicações paralelas e distribuídas. A Seção 1.4. apresenta as principais técnicas de análise utilizadas para se melhorar o desempenho das aplicações. A Seção 1.5. apresenta algumas ferramentas de coleta e análise. A Seção 1.6. traz uma conclusão a este texto.

1.2. Conceitos básicos

Existem vários conceitos que influenciam diretamente as fases de coleta e de análise. Por exemplo, o problema do não-determinismo do sistema paralelo tem um impacto direto na análise de comportamento pois não necessariamente um comportamento anômalo aparece em todas as execuções da aplicação paralela para a mesma entrada. Esta seção apresenta alguns conceitos básicos sobre análise de desempenho de programas paralelos: efeito de sonda e intrusão, análise em tempo de execução (*online*) e após a execução (*offline* ou *post-mortem*), e sincronização de relógios. O final da seção traz um sumário de outros conceitos básicos.

1.2.1. Efeito de sonda

Uma aplicação paralela, quando executada sem nenhum tipo de coleta de dados, demonstra um **comportamento natural**. Ele é caracterizado pela ausência de qualquer outra atividade que não seja relacionada diretamente com os objetivos da aplicação.

Qualquer processo de análise de comportamento de uma aplicação paralela passa obrigatoriamente por uma fase de coleta de dados. Nesta fase, deve-se registrar informações consideradas pertinentes para se entender o comportamento da aplicação. Existem várias técnicas diferentes, que serão vistas na Seção 1.3., que podem ser utilizadas para realizar o registro dessas informações comportamentais. Independentemente de qual técnica é utilizada, o simples fato de registrar qualquer informação causa um impacto no comportamento da aplicação sendo monitorada.

O tempo gasto no registro do comportamento da aplicação é chamada de **efeito de sonda**, ou seja, a intrusão causada pela sonda de monitoramento inserida dentro da aplicação para registrar o seu comportamento. Este impacto pode ocorrer pela utilização de diferentes tipos de recursos: memória, utilização de CPU, disco ou qualquer outro recurso computacional necessário para se registrar o comportamento.

A adoção de um sistema de coleta específico para uma aplicação paralela deve levar em conta o efeito de sonda para o conjunto de informações que se deseja coletar. Procura-se sempre que o efeito de sonda, ou seja o impacto causado pela coleta, seja o menor possível. Em geral, o impacto é medido através de uma porcentagem de tempo em relação ao comportamento normal da aplicação. Por exemplo, pode-se dizer que utilizando um determinado sistema de coleta obtém-se um impacto de 3% do tempo de execução. Essa medida permite comparar vários sistemas de coleta levando a escolha daquele que tem o menor impacto medido em relação a quantidade de informação coletada.

Muitas vezes a escolha do sistema de coleta deve ser feita levando em conta outros fatores correlacionados com o efeito de sonda. Embora o tempo seja o mais usado, a ocupação da memória, processamento e disco podem ser determinantes em sistemas com esses recursos limitados. Por exemplo, em um sistema paralelo com pouca memória disponível talvez seja desejável escolher um sistema de coleta que minimiza a ocupação de memória.

Caracterização da intrusão

Quantificar e caracterizar o efeito de sonda diz quão diferente está o comportamento registrado em relação ao comportamento natural da aplicação, quando nenhum tipo de observação é feito em tempo de execução. Esta medida da intrusão pode ser utilizada para corrigir o comportamento alterado registrado pelo sistema de coleta, tentando se aproximar ao máximo do comportamento natural da aplicação. No melhor caso, o objetivo da compensação da intrusão é anular totalmente o efeito de sonda causado pelo sistema de coleta.

Intrusão zero

Aplicações paralelas executadas em sistemas paralelos reais tem seu comportamento natural alterado no momento que elas são observadas para serem analisadas. Uma possibilidade para lidar com este problema é melhorar as ferramentas de coleta de forma que seu impacto se torne o menor possível, combinando isto com alguma técnica sofisti-

cada de compensação da intrusão. Mesmo com boas medidas de compensação do efeito de sonda, é difícil anular completamente a intrusão causada pela observação.

Uma alternativa para se obter impacto zero na observação do comportamento é utilizar simuladores de sistemas paralelos. Executando as aplicações paralelas sobre estes simuladores, o desenvolvedor é capaz de registrar qualquer informação que ele achar pertinente para a análise sem nenhum tipo de intrusão. Isso advém do fato que no momento da coleta e do registro da informação propriamente dito, a simulação está parada. Os simuladores construídos com o arcabouço SimGrid [CAS 2008] exploram esta característica e permitem registrar o comportamento de qualquer simulação com impacto zero no comportamento da aplicação simulada.

1.2.2. Análise *online* e *offline*

Como visto anteriormente, a análise de desempenho de uma aplicação paralela é dividida em duas fases: a de coleta e a fase de análise propriamente dita. Existem duas formas de combiná-las: a abordagem *online* realiza tanto a coleta quanto o procedimento de análise simultaneamente, em tempo de execução; a abordagem *offline*, também conhecida como *post-mortem*, realiza as duas fases separadamente, sendo que a análise ocorre sempre após o fim da execução da aplicação cujo comportamento será analisado. As vantagens e desvantagens de cada uma das abordagens são discutidas a seguir, seguidas por uma discussão a respeito da aplicação de cada uma das técnicas.

Análise *online*

Realizar simultaneamente a coleta e análise de dados comportamentais traz uma série de vantagens dentre as quais a ausência do custo de manutenção de dados e interatividade na análise dos dados. A ausência do custo de manutenção de dados significa que é desnecessário registrar os dados em disco, uma vez que os mesmos são analisados em tempo de execução. Quando a aplicação paralela a ser analisada é grande, em quantidade de processos, e longa, os dados a serem registrados podem ser substanciais. A interatividade na análise dos dados significa que o analista pode influenciar os rumos do processo de análise durante a execução da aplicação. Por exemplo, o analista pode iniciar a análise *online* utilizando um conjunto de índices de desempenho relativamente grande. A medida que a execução prossegue e eventuais problemas de desempenho surgem, o analista pode alterar os índices para selecionar outros que permitam realizar uma investigação mais profunda em somente uma parte do programa paralelo.

As desvantagens da análise *online* estão relacionadas com a falta de escalabilidade e o custo da transferência dos dados. A quantidade de informação a ser analisada em tempo de execução aumenta de acordo com o tamanho das aplicações paralelas a serem analisadas. Atualmente, as aplicações podem ser compostas de dezenas de milhares de processos. O custo de realizar a análise desses dados ao mesmo tempo que a aplicação se executa se manifesta de diferentes formas, tanto na utilização do processador quanto na rede de interconexão do sistema paralelo. O processador se torna necessário não só para a execução da própria aplicação, mas também para executar as tarefas de análise dos dados comportamentais que descrevem o desempenho da aplicação. A rede de interconexão é utilizada de forma contínua para transferir os dados coletados, eventualmente centralizando-os para analisá-los. Essa utilização concorrente dos recursos computacionais pode aumentar o efeito de sonda (veja Seção 1.2.1.) quando a aplicação é

muito grande, eventualmente tornando impraticável a análise.

Poucas ferramentas implementam uma análise puramente em tempo de execução. Os problemas de escalabilidade da técnica e a alteração substancial do comportamento natural do programa a ser analisado são as principais razões da escolha por outras técnicas em detrimento da análise *online*. Em virtude disso, a técnica mais utilizada atualmente é a análise *offline*, descrita a seguir.

Análise *offline*

As vantagens e desvantagens da análise *offline*, ou *post-mortem*, são opostas àquelas da análise *online*. Por exemplo, enquanto que na análise em tempo de execução o custo de manutenção dos dados é reduzido, a abordagem *offline* requer bastante esforço para o gerenciamento desses dados. Uma aplicação paralela com dezenas de milhares de processos pode gerar uma quantidade significativa de dados que devem ser registrados em disco. Normalmente, uma análise *offline* não permite interatividade durante a fase de observação e registro do comportamento da aplicação paralela. Isso requer do analista um bom planejamento da bateria de experimentos que deverá ser feita para se extrair informações úteis. Isso pode ser especialmente problemático em aplicações de longa duração.

A principal vantagem da abordagem *offline* está relacionada com escalabilidade. Uma cuidadosa seleção de técnicas de observação e registro habilita o uso da abordagem em aplicações de larga escala. Essas técnicas de baixa intrusão incluem utilização de espaço em memória durante a observação para evitar utilização excessiva do disco, diminuindo o efeito de sonda ao máximo, e também o uso de formatos de dados binários com compactação.

Aplicabilidade das análises *offline* e *online*

Cada tipo de análise visto nesta seção é apropriada para um tipo de estudo a respeito de uma aplicação paralela. A observação em tempo de execução da abordagem *online* permite uma análise da aplicação mais relacionada com a atividade de depuração de erros, similar a depuração de programas sequenciais, do que com a análise do desempenho propriamente dito. A abordagem *post-mortem* permite uma análise focada exclusivamente no desempenho, visto que seu objetivo principal é causar a menor intrusão possível. Por conta disso, a maioria das ferramentas recentes de análise de desempenho como Pajé [SCH 2012] e seu arcabouço de ferramentas, Vampir [BRU 2010], Vite [COU 2009] e Scalasca [GEI 2010] fazem a análise em duas fases separadas.

1.2.3. Sincronização de relógios

A sincronização de relógios é fundamental em sistemas distribuídos e paralelos. Através dela, a noção de relógio global pode ser atingida de forma a manter a consistência dos dados observados a respeito dos vários processos da aplicação paralela a ser analisada. Essa consistência se manifesta principalmente nas situações onde a relação de causalidade deve ser obedecida.

A Figura 1.1 mostra dois exemplos de observação que ilustram a necessidade da sincronização de relógios. A flecha pontilhada representa o envio de uma mensagem do processo A para o processo B, cada um executando em um nó computacional diferente do sistema paralelo. Cada processo registra de forma independente o envio e a recepção da

mesma mensagem. Na situação da esquerda, sem sincronização de relógios, o processo que envia registra que o envio da mensagem aconteceu no instante 10.2, enquanto que o receptor registra a recepção da mesma mensagem no instante 9.5. Sem sincronização de relógios, durante o período da coleta, as informações são registradas sem nenhum problema. É somente após a execução que o analista vai perceber a inconsistência, ou seja, uma mensagem não pode ser recebida antes de ser enviada. Na situação da direita, desta vez com sincronização de relógios, a situação é remediada no momento da observação dos dados, onde se pode inclusive chegar a conclusão que o tempo de transferência da mensagem foi de 0.3 unidades de tempo.

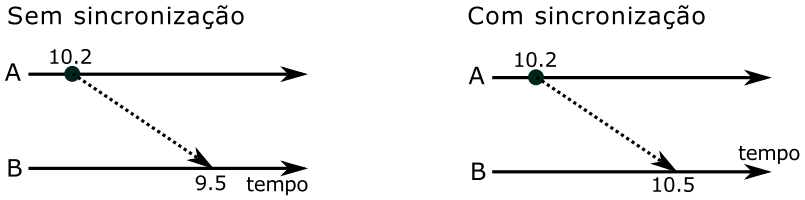

Figura 1.1: Exemplo ilustrando a necessidade de sincronização de relógios.

Embora fundamental, realizar a sincronização de relógios não é uma tarefa fácil. Vários algoritmos existem, grande parte deles oriundos do trabalho de Lamport [LAM 78], inclusive com simplificações que permitem pelo menos manter a ordem parcial entre os eventos observados de uma aplicação.

O protocolo mais difundido de sincronização de relógios é NTP (*Network Time Protocol*), organizando hierarquicamente e capaz de sincronizar relógios na ordem dos milisegundos. Embora difundido, o resolução da sincronização de relógio do protocolo NTP é insuficiente para a análise de aplicações paralelas. Estando na ordem dos milisegundos, as redes de alto desempenho que interligam os supercomputadores são capazes de transferir grandes mensagens em tempos significativamente menores que um milisegundo. Nesta situações, o protocolo NTP, mesmo nas condições ideais de execução, é incapaz de fornecer uma garantia de sincronização de relógio suficiente para uma análise coerente do comportamento de aplicações paralelas.

O PTP (*Precision Time Protocol*), ou protocolo de tempo preciso, foi proposto pela IEEE e em redes de interconexão com baixa latência, o protocolo é capaz de sincronizar relógios na ordem de microsegundos. Isso o torna utilizável na análise de desempenho de aplicações paralelas.

1.2.3.1. Sincronização de relógios por *hardware*

Outras soluções existem baseadas em equipamentos específicos que são acoplados aos sistemas computacionais paralelos para fornecer uma hora precisa. O exemplo mais clássico é o uso de relógios atômicos, capazes de fornecer uma hora extremamente precisa. O problema dessa abordagem é o custo: evidentemente que acoplar um relógio atômico a cada computador do sistema computacional paralelo é impraticável. Uma alternativa mais barata é a utilização de receptores GPS (*Global Positioning System*), visto que cada satélite GPS é equipado com vários relógios atômicos que contribuem para a precisão do tempo GPS. Essa precisão é refletida aos receptores que podem então ser utilizados pelos sistemas computacionais como referência de relógio. A precisão deste tipo

de equipamento é na ordem dos nanosegundos, largamente suficiente para a medição e análise de aplicações paralelas.

1.2.4. Outros conceitos

Existem outros conceitos básicos relacionados a análise de desempenho que são igualmente importantes. Dentre eles, destacam-se a execução não-determinística de aplicações paralelas e a obtenção de um estado global da aplicação.

A execução não determinística diz respeito ao fato que a mesma aplicação paralela, quando executada com o mesmo conjunto de dados de entrada, pode apresentar comportamentos diferenciados de acordo com influências externas a aplicação. Por exemplo, a ordem de recepção nas mensagens trocadas entre os processos não necessariamente é a mesma entre duas ou mais execuções idênticas. Esta característica inerente de sistemas paralelos e distribuídos torna difícil a análise de desempenho, pois um determinado comportamento considerado ruim não necessariamente aparece em todas as execuções da aplicação. Uma possibilidade para se controlar o indeterminismo é reexecutar a aplicação paralela de forma determinística, após uma primeira execução cuja ordem das ações é registrada.

A obtenção de um estado global da aplicação (*snapshot*) é particularmente útil pois ele permite verificar propriedades globais considerando todos os processos envolvidos na aplicação. No entanto, obter um estado global da aplicação é considerado uma tarefa complexa devido a latência da rede de interconexão e a eventual intrusão causada no desempenho da aplicação.

Os conceitos de indeterminismo na execução de aplicações paralelas, obtenção de estado global entre outros já foram apresentados em um minicurso da Escola Regional de Alto Desempenho, na sua primeira edição [STE 2001].

1.3. Técnicas de observação e registro de comportamento

A primeira fase da análise de desempenho consiste na observação e registro do comportamento do programa paralelo que será posteriormente analisado, supondo um procedimento de análise *post-mortem*. Esta seção apresenta os diferentes conceitos de técnicas de observação diferentes assim como as técnicas de coleta de dados de comportamento de aplicações paralelas.

1.3.1. Técnicas de observação

A observação diz respeito aos métodos utilizados para observar o comportamento do programa, eles podem ser tanto internos quando externos. A observação interna é feita através da inserção de sondas no próprio programa a ser analisado, enquanto que na observação externa apenas o efeito do programa paralelo no sistema de execução é registrado. Existe uma variedade de técnicas de observação, tais como:

Monitoramento é uma técnica de observação externa baseada em mecanismos já existentes no sistema computacional. A principal vantagem do monitoramento é que o

programa ou sistema a ser observado não precisa ser alterado para que seu comportamento seja estudado. Em geral, o monitoramento é utilizado para a observação de sistemas computacionais e suas redes de interconexão.

Geração de índices estatísticos é uma técnica de observação interna que permite observar as tendências estatísticas do comportamento do programa paralelo através do cálculo de métricas de desempenho de diferentes tipos, tais como o tempo de espera em certas regiões do programa, o tempo médio de execução de uma tarefa, entre outros. O programador pode utilizar tais métricas para isolar uma parte do programa que precisa ser melhorada. Ela necessita a utilização de alguma técnica de coleta de dados (veja a Seção 1.3.2.).

Definição de um perfil de execução é uma técnica de observação interna ou externa que consiste a estimar o tempo gasto em cada uma das partes do programa paralelo. Uma parte do programa pode ser uma região de código, uma função, um método ou um módulo do programa. Através do perfil de execução, o programador identifica facilmente qual parte do programa utiliza mais tempo para ser executado. No entanto, um perfil é sempre um indicador global e deve ser utilizado com cautela em execuções longas.

Observação comportamental é uma técnica de observação interna capaz de indicar exatamente quanto tempo uma parte do código precisou para ser executada e, diferentemente da técnica de perfil, dizer exatamente em que momento da execução isso ocorreu. É através da observação comportamental que o analista é capaz de ter todas as informações necessárias para uma análise de qualidade. Essas informações podem ser tanto globais (como o perfil de execução) quanto locais, no tempo e no espaço. Em geral, a análise de desempenho mais sofisticada é realizada utilizando observação comportamental.

Qual a melhor técnica de observação?

A melhor técnica de observação depende do tipo de análise de desempenho que se deseja realizar e da quantidade de conhecimento que o desenvolvedor tem do comportamento da sua aplicação paralela. Por exemplo, em um primeiro momento, o analista pode gerar um perfil de execução para atacar o principal problema de desempenho. Em etapas subsequentes de análise, pode-se realizar uma observação comportamental para se detectar problemas de desempenho esporádicos. Em suma, o analista deve escolher a técnica de observação que melhor se adapta tanto à aplicação quanto ao sistema paralelo que será utilizado para a execução.

1.3.2. Técnicas de coleta e registro

A situação ideal para um sistema computacional de coleta e registro de comportamento é que ele seja preciso, tenha um nível de intrusividade próximo de zero, com qualquer tipo de efeito de sonda passível de compensação. Esta situação ideal é naturalmente difícil de obter devido as características dos sistemas computacionais. Mesmo assim, uma grande quantidade de técnicas de registro e coleta surgiram com diferentes equilíbrios entre precisão e intrusividade.

A Figura 1.2 classifica os principais métodos de coleta de acordo com a maneira que o registro da informação é lançado: pelo tempo ou por eventos [REE 94]. As técnicas dirigidas pelo tempo coletam as informações periodicamente, entre intervalos regulares de tempo especificado pelo analista. A técnicas mais representativa desta abordagem é a amostragem. Quando a coleta do comportamento é dirigido por eventos, o registro da informação é disparado pela ocorrência de eventos inseridos no código da aplicação pelo analista ou por alguma biblioteca de observação. As técnicas guiadas por eventos podem ser contagem e rastreamento. Nesta seção são detalhadas rapidamente cada uma dessa técnicas.

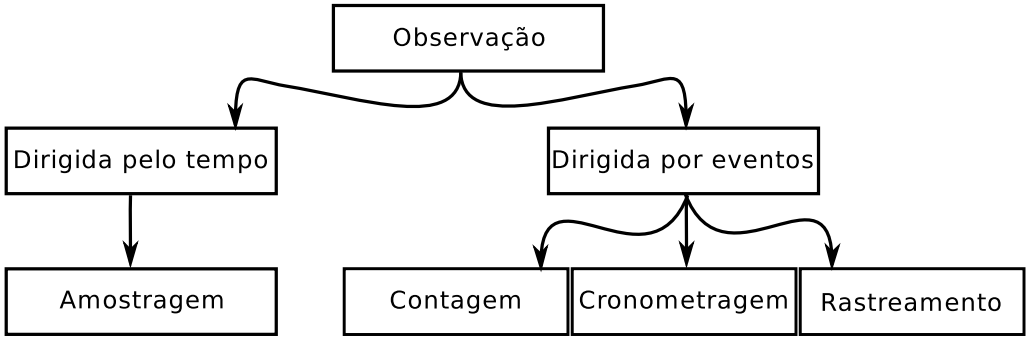

Figura 1.2: Classificação das técnicas de coleta de dados comportamentais.

1.3.2.1. Amostragem

A técnica de amostragem consiste em examinar periodicamente o estado de um programa ou o valor de uma variável necessária para a análise de desempenho. A medição do valor acontece a intervalos regulares de tempo, definidos pelo analista, estabelecendo assim a frequência da amostragem. No momento que deve ocorrer a medição, o sistema de coleta dispara uma ordem de observação do comportamento da aplicação. O sistema se reconfigura para que a medição ocorra novamente após o intervalo de tempo definido.

Um caso bastante comum de uso da técnica de amostragem é a geração de um perfil da aplicação. Por exemplo, o analista pode definir que a cada vinte microsegundos o sistema deve verificar qual função do programa está sendo executada e registrar essa informação em disco ou em memória, incrementando um contador associado ao nome da função. No final da execução, o analista tem acesso ao perfil de execução do programa com a lista de funções cuja execução foi mais detectada pelo sistema de amostragem.

O perfil gerado pela amostragem é capaz de dizer somente quantas vezes uma função foi executada segundo a amostragem. Embora seja possível derivar uma duração aproximada a partir da quantidade de observações e o intervalo de amostragem, as ferramentas de análise em geral mostram os valores como uma porcentagem das amostras.

A qualidade da amostragem é inversamente proporcional ao tamanho do intervalo de tempo entre cada amostra. Quanto menor for o intervalo, maior será a qualidade da amostra. Se o intervalo de amostragem for de 100 nanosegundos, por exemplo, é pouco provável que algum comportamento da aplicação escape ao processo de amostragem. Isso acontece porque em geral as funções de uma aplicação paralela se executam na ordem dos microsegundos. Em contrapartida, quanto maior for o intervalo, menor será a confiabilidade do perfil obtido no final da execução da aplicação. Por exemplo, caso o intervalo de

amostragem seja muito grande, existem grandes chances que comportamentos que duram menos que o intervalo estabelecido não sejam capturados por amostragem.

O efeito de sonda é outro aspecto importante na definição do intervalo de amostragem. Caso a frequência de amostragem seja muito alta, a intrusão causada pela coleta será excessiva. O analista deve definir a frequência de amostragem através de um equilíbrio entre uma frequência de amostragem adequada, capaz de identificar os comportamentos relevantes da aplicação, e o nível de intrusão. Encontrar um bom equilíbrio entre estes dois fatores é essencial para que a análise seja de qualidade.

1.3.2.2. Cronometragem

A cronometragem consiste em medir o tempo passado em uma região do código da aplicação paralela através da inserção de instruções extras responsáveis pela cronometragem de tempo, implementando portanto uma técnica de observação interna. Da mesma forma que a técnica de amostragem, a cronometragem é capaz de gerar um perfil da execução do programa paralelo levando em conta as funções ou regiões do código que são cronometradas em tempo de execução. A diferença em relação a amostragem é que esta técnica é dirigida pelas instruções adicionadas pelo analista no programa a ser analisado, sendo portanto guiada por eventos e não pelo tempo. Outra diferença é que a cronometragem mede exatamente quanto tempo é gasto nas funções, ao contrário da amostragem que mede somente a frequência no qual as funções são detectadas em execução.

A intrusão causada pela técnica de cronometragem está relacionada diretamente com a quantidade de modificações feita pelo desenvolvedor para cronometrar o tempo passado em regiões e funções da aplicação. A intrusão da cronometragem é portanto mais controlável quando comparada a técnica de amostragem, pois o desenvolvedor pode selecionar somente um número reduzido de funções para cronometrar. No entanto, caso os pontos de cronometragem sejam executados muito frequentemente, é provável que, mesmo com poucos pontos de coleta, o efeito de sonda seja significativo. Isso pode ocorrer, por exemplo, se o analista decidir cronometrar o tempo de execução de cada laço de repetição através da inserção de cronômetros dentro do laço. Caso a repetição ocorra milhares de vezes, ocorrerá uma alteração excessiva do comportamento natural do programa fazendo com que o perfil de execução gerado seja de baixa qualidade.

O problema da técnica está relacionada a sua vantagem. Para implementar a cronometragem em uma aplicação paralela, o desenvolvedor é obrigado a alterar o código da sua aplicação ou utilizar alguma ferramenta de instrumentação para cronometrar regiões de código. Dependendo da complexidade da aplicação, pode ser difícil definir os pontos de instrumentação que potencializem uma boa qualidade da informação comportamental coletada e do perfil de execução gerado no fim da execução.

1.3.2.3. Contagem

Um sistema de coleta por contagem é um sistema guiado por eventos que permite medir a quantidade de vezes que um determinado comportamento ocorre, como a execução de uma função, o acesso a uma variável compartilhada, a um recurso. Através de contadores, o analista pode descobrir qual parte do programa é mais utilizada e então focar seus esforços de otimização nesta região.

A maior vantagem da contagem é a sua baixa intrusão. O efeito no comportamento natural da aplicação paralela sendo analisada se resume a um incremento em um contador

de utilização diretamente relacionado ao evento que aconteceu. Devido a isto, a contagem pode ser utilizada em aplicações que tem longa duração e nos eventos que potencialmente são muito frequentes.

A implementação da técnica de contagem pode ser feita em diferentes níveis do sistema computacional, tanto pelo intermédio de programas de computadores, tanto no nível do equipamento computacional. A vantagem de se implementar contagem utilizando programas é a flexibilidade, a capacidade de adaptação e portabilidade. Essas boas características em geral não estão presentes em contadores embutidos em equipamentos, pois sofrem de imutabilidade. A desvantagem da implementação de contadores por programas é que o efeito de sonda pode ser mais significativo quando comparada a uma implementação direta no equipamento computacional.

Existe uma série de contadores já embutidos nos processadores atuais. Eles são conhecidos como contadores de *hardware*, capazes de medir diferentes aspectos do funcionamento de um processador tais como a quantidade de faltas no acesso a memória cache, a quantidade de instruções executadas, a quantidade de execuções de uma instrução em particular, e assim por diante. Devido a grande diversidade de arquitetura de processadores e funcionalidades, existe um esforço de padronização do acesso aos valores dos contadores através de interfaces tais como PAPI [BRO 2000].

1.3.2.4. Rastreamento

O rastreamento é uma forma de coleta interna que tem por objetivo o registro de informações consideradas significativas a respeito do comportamento do programa e que possam ser utilizadas para reconstruir o comportamento original do programa para fins de análise. Essas informações ficam registradas em arquivos de rastro, onde cada arquivo de rastro é composto por uma série de eventos normalmente ordenados pelo tempo. Através da totalidade de arquivos de rastro e seus respectivos eventos, o analista deve ser capaz de reconstruir o estado da aplicação tal qual ele era no momento da execução. A análise de desempenho se faz sob esta visão do comportamento do programa, derivada a partir dos eventos registrados durante a execução.

Um evento é a ocorrência de um comportamento específico ao longo da execução do programa a ser analisado. Ele é composto por um tipo e uma data, seguido por informações relacionadas ao tipo, cada qual registrada sob a forma de um campo de dado. Exemplos de eventos podem ser o início da execução de uma função, o acesso a um recurso de memória, ao envio de uma mensagem, etc.

O rastreamento através de eventos é necessário para a reconstrução do comportamento original do programa paralelo após a sua execução. Essa reconstrução deve ser a mais precisa e coerente possível, obedecendo a ordem causal entre os eventos e o ordenamento temporal registrado nos rastros. Quanto mais preciso for o rastreamento do programa, com uma maior quantidade de eventos, mais fiel será a reconstrução do comportamento a partir dos rastros. A melhor situação do ponto de vista da reconstrução é que todo e qualquer tipo de evento possa ser registrado durante a execução. Isso é impraticável pois gerará muita intrusão durante a coleta, alterando demasiadamente o comportamento natural do programa. Encontrar a precisão ótima do rastreamento é portanto uma otimização com múltiplos objetivos: ela deve maximizar a quantidade de eventos registrada ao mesmo tempo que minimiza a intrusão causada pelo registro de eventos.

Uma série de informações são necessárias para a reconstrução do comportamento do programa. Além da data do evento indicando o momento de sua ocorrência, dados

a respeito das entidades computacionais envolvidas na geração do evento são também necessárias. Uma entidade pode ser tanto um processo, um fluxo de execução (*thread*) ou qualquer outra entidade cujo comportamento é registrado ao longo da execução. Por exemplo, o evento de recepção de mensagem deve conter o identificador da entidade que enviou a mensagem assim como da entidade que a recepcionou.

Em posse dos rastros, o analista pode realizar vários tipos de análise (ver Seção 1.4. a seguir), sendo portanto a técnica mais geral de coleta de dados comportamentais. Com os eventos é possível, por exemplo, derivar perfis de execução e criar histogramas da mesma forma que é possível com técnicas de amostragem e cronometragem. A técnica de contagem pode ser também substituída por rastreamento através da utilização de eventos que registram os valores dos contadores. A principal diferença está na caracterização temporal dos valores: enquanto que nas outras técnicas somente valores absolutos obtidos no final da execução estão disponíveis, no rastreamento os dados são correlacionados com o tempo.

Formas de rastreamento Existem três tipos de rastreamento possíveis em sistemas computacionais paralelos: rastreamento por *hardware*, por *software* e híbrido. O primeiro deles, por *hardware*, utilizam equipamentos especializados encarregados de registrar os eventos necessários para a reconstrução do estado da aplicação paralela. Embora com baixa intrusão, qualquer implementação por *hardware* tem baixa flexibilidade, sendo difícil de alterar. O rastreamento por *software* implementa todo o rastreamento no nível da aplicação paralela ou nas suas bibliotecas associadas. Isto é feito através de instruções extras embutidas no programa ou disponíveis nas bibliotecas capazes de gerar eventos, registrando-os nos momentos oportunos em disco para posterior análise. Por fim, o rastreamento híbrido é uma mescla de rastreamento por *software* e *hardware*, sendo empregado nas situações onde um bom equilíbrio entre os dois tipos de rastreamento pode ser definido.

Com exceção de casos bem específicos, a maioria das ferramentas (veja Seção 1.5.) implementa rastreamento somente por *software*, devido a flexibilidade e a constante evolução dos sistemas computacionais paralelos.

Qual a melhor técnica de coleta e registro?

A melhor técnica de coleta e registro depende de qual nível de conhecimento sobre o comportamento da aplicação pretende-se obter. A técnica de amostragem pode ser escolhida caso o analista necessite de um panorama geral do comportamento da aplicação paralela, sem a necessidade de alterar o código. Caso seja necessário um perfil com um histograma temporal mais preciso, pode-se escolher a técnica de cronometragem embutida nas funções que o analista suspeita serem as mais custosas. A análise pode ser enriquecida com contadores de *hardware* para correlacionar o comportamento no nível aplicativo com o seu impacto no sistema. Por fim, pode ser aplicada uma técnica de rastreamento para detectar problemas de comunicação entre os processos.

Dependendo da situação e do conhecimento prévio do analista a respeito do comportamento do programa paralelo, é possível que somente uma técnica de coleta e registro seja suficiente para as necessidades de otimização de desempenho do programa. Neste caso, a escolha de qual método de coleta utilizar deve estar relacionada a suposição feita pelo desenvolvedor a respeito do programa, utilizando os dados apenas para confirmar ou não esta suposição.

1.4. Técnicas para análise de desempenho

Existem várias técnicas de análise de desempenho para aplicações paralelas e distribuídas, a grande maioria inspiradas diretamente pelas formas de coleta de dados comportamentais vistas na Seção 1.3.. O analista pode, por exemplo, realizar a análise através de índices estatísticos calculados a partir de dados obtidos por contadores, ou ainda construir uma representação visual da informação coletada através de rastreamento. Essa variabilidade de técnicas para análise de desempenho pode ser vista como complementar: enquanto algumas técnicas dão uma visão global do desempenho do programa, outras são capazes de explorar pontos bem locais do comportamento da aplicação paralela.

A metodologia de análise adotada pelo analista depende diretamente da natureza das questões de desempenho que estão sob investigação. Questões relacionadas com as características da aplicação, se ela é limitada pela CPU ou pela rede de interconexão, entre outras, devem ser levadas em conta e ser utilizadas para selecionar a melhor técnica de análise de desempenho para cada caso. Mesmo que não exista uma solução única para a análise, uma abordagem simples é obter um visão geral do comportamento da aplicação, e depois procurar por detalhes e questões de desempenho menores. A visão geral pode ser obtida com a geração de um perfil de execução, através de amostragem ou cronometragem, através de uma medição direta ou indireta. Depois da resolução dos problemas detectados por esta técnica, o analista pode obter um visão mais detalhada do desempenho através do rastreamento da aplicação com pontos de registro de comportamento bem escolhidos. Os rastros de uma aplicação paralela em geral são centralizados para permitir uma análise interativa ou automática dos dados.

Os objetivos da análise de desempenho de uma aplicação paralela são os seguintes:

Melhorar o desempenho da aplicação paralela O desempenho de uma aplicação paralela pode ser medido através de diversos fatores correlacionados dentre os quais: tempo de execução, aceleração, eficiência. As técnicas de análise de desempenho devem ser capazes de dar informações que possam ser utilizadas para melhorar esses índices de desempenho.

Aumentar a eficiência de utilização dos recursos A utilização dos recursos computacionais de forma eficiente é fundamental na execução de uma aplicação paralela. A análise de desempenho deve fornecer meios ao analista de identificar uma baixa eficiência no uso dos recursos que pode, juntamente com outros fatores, ser a razão de um baixo desempenho da aplicação.

Esta seção apresenta algumas técnicas de análise de desempenho de aplicações paralelas que podem ser utilizadas tanto em nível global quanto local. Da mais simples a mais complexa, serão vistas a análise através de perfis de execução, a análise automática, a análise por transformação de dados e finalmente a análise interativa através da visualização de rastros.

1.4.1. Análise de perfis de execução

A análise utilizando perfis de execução é a forma mais simples para se obter uma primeira ideia a respeito do comportamento da aplicação paralela. Ela consiste em analisar a aplicação através de um histograma que lista as funções cuja execução foi mais frequentemente detectada. Este histograma pode refletir o comportamento da aplicação

paralela inteira, considerando todos os processos envolvidos, e do comportamento de cada processo, de forma individual.

Existem várias formas para se obter o perfil de execução de uma aplicação paralela. O uso das técnicas de amostragem e cronometragem podem ser utilizadas para se obter as informações para a criação de um perfil de execução. Como discutido anteriormente (veja Seção 1.3.2.1.), o analista deve estar ciente da frequência de amostragem para obter um perfil de execução que condiz com a realidade. O rastreamento também pode ser utilizado para se gerar um perfil de execução. No entanto, neste caso o analista deve criar o histograma de execução a partir dos rastros gerados pela aplicação paralela.

1.4.2. Análise automática

Os supercomputadores atuais são compostos de dezenas de milhares de processadores, sendo explorados por aplicações com uma quantidade de processos cada vez maior. Quando alguma técnica de rastreamento do comportamento é utilizada, o resultado é uma imensa quantidade de dados a serem analisados. A análise manual destes dados pelo analist, através de uma abordagem exploratória e interativa, pode ser difícil de ser realizada principalmente devido ao volume dos dados.

A análise automática de dados comportamentais consiste em detectar problemas de desempenho de aplicações paralela de forma automática, sem a interação do analista. Para tal, ela utiliza como entrada o registro de comportamento de aplicações paralelas coletadas pelas técnicas vistas na Seção 1.3.. A técnica de rastreamento é a mais utilizada pois esta fornece, através dos rastros, um detalhamento maior do comportamento da aplicação paralela. No momento que um problema de desempenho é identificado, o desenvolvedor interage com a ferramenta para tentar encontrar a razão do problema.

A maior vantagem da análise automática é a escalabilidade. A técnica sofre menos na situação de uma grande quantidade de dados a serem analisadas quando comparada a uma análise tradicional, com intervenção do analista onde este deve analisar manualmente o conjunto de dados. A escalabilidade se materializa principalmente na análise de aplicações potencialmente grandes, com milhares de processos comunicantes. Nestes casos, a análise automática pode ser vista como uma ferramenta de mineração de dados onde o objetivo é encontrar problemas de desempenho e listá-los ao desenvolvedor. A tarefa de mineração de dados pode ser paralelizada e distribuída na própria infraestrutura paralela que é utilizada para a execução da aplicação paralela cujo comportamento é analisado. Esta paralelização contribui diretamente para a maior escalabilidade da abordagem, considerando que uma análise manual, por exemplo, não pode ser paralelizada visto que na maioria dos casos existe somente um analista.

Um conjunto de problemas de desempenho devem ser previamente conhecidos para que a análise automática possa funcionar. Estes problemas são descritos sobre a forma de padrões de comportamento que a mineração de dados deve procurar sobre os rastros da aplicação paralela a ser analisada. Cada padrão contém a quantidade de processos envolvidos e qual o comportamento de cada processo (estados e valor de variáveis), inclusive com o estado da comunicação entre os processos. Os diferentes padrões de mau desempenho podem ser classificados por tipo e pela quantidade de processos envolvidos.

A desvantagem da análise automática está relacionada ao conjunto de padrões de baixo desempenho definidos previamente. Independente de quão sofisticada é a mineração de dados, a análise automática sempre será limitada aos padrões conhecidos. Caso a aplicação paralela manifeste um baixo desempenho e o padrão correspondente não tenha

sido previamente definido, o problema passará despercebido pela análise.

1.4.3. Análise por transformação de dados

Uma técnica de análise desempenho possível é a transformação dos dados oriundos da fase de coleta e registro do comportamento de uma aplicação paralela. A análise de desempenho utiliza como entrada então os dados transformados e não os dados originais que foram coletados. Várias razões existem para se implementar a análise desta forma. A primeira delas é a necessidade de reduzir o volume de dados originais coletados para que eles sejam compreensíveis. Isso é especialmente útil para aplicações de larga escala compostas de milhares de processos cujo comportamento é, por exemplo, rastreado. A segunda razão é a necessidade de criação de novas métricas de desempenho, úteis para a análise, mas que estão indisponíveis nos dados originais.

A análise de desempenho por transformação de dados consiste em alterar a natureza das métricas comportamentais de forma a colocar em evidência características relevantes do comportamento da aplicação paralela. Por exemplo, em aplicações de larga escala com muitos processos, é possível que grande parte dos processos se comportem de maneira muito parecida. Nestas situação, a técnica de análise por transformação pode utilizar algum algoritmo auxiliar de análise que agrupa processos cujo comportamentos são parecidos.

A transformação de dados se aplicar melhor em dados oriundos de rastreamento. Esta técnica de coleta, sendo a mais abrangente, coleta uma infinidade de detalhes temporais a respeito do comportamento do programa, desde estados da aplicação até contadores de *hardware*. Ela oferece, portanto, maiores possibilidades de transformação quando considerado os dados oriundos de outras técnicas de coleta e registro como amostragem ou cronometragem. A análise por transformação pode ser classificada em duas abordagens: agregação e agrupamento.

Agregação Os algoritmos de agregação de dados tem por objetivo agregar as informações comportamentais de diferentes entidades computacionais de uma aplicação paralela, utilizando um operador de agregação relevante para a análise. O resultado final da agregação depende diretamente de qual operador é utilizado. Por exemplo, o comportamento médio das entidades agregadas será obtido caso um operador de média seja utilizado; o pior comportamento será escolhido caso um operador de mínimo seja adotado, e assim por diante.

Para dados oriundos de amostragem ou cronometragem, a agregação pode ser feita diretamente sobre os perfis de execução obtidos de cada processo de uma mesma aplicação paralela. No caso de dados oriundos de rastreamento, a agregação pode ser tanto espacial quanto temporal. A agregação temporal diz respeito a integração realizada no tempo de variáveis e métricas de um único processo ou fluxo de execução. No caso da agregação espacial, ela é realizada utilizando como entrada o comportamento de mais de um processo ou fluxo de execução.

Agrupamento (*Clustering*) Os algoritmos de agrupamento [LEE 2008, JOS 2006], algumas vezes hierárquicos [AGU 2006], tem por objetivo agrupar processos cujo comportamento é similar de acordo com uma ou mais métricas disponíveis a respeito dos processos. Uma vez definidos os grupos, escolhe-se um processo para

representar o comportamento de todos os processos que foram agrupados. A escolha desse processo representativo é feita minimizando as diferenças entre esse processo e todos os outros do grupo.

No caso de dados oriundos de amostragem ou cronometragem, o agrupamento pode ser aplicado para escolher quais processos de uma mesma aplicação paralela tem perfis de execução semelhantes. Os grupos criados podem ser então analisados procurando por grupos cujo comportamento representativo difere demasiadamente daquele esperado. Para a análise de rastros de execução, o agrupamento pode ser utilizado para detectar processos cujo comportamento é semelhante ao longo do tempo. Isto leva a uma redução da complexidade da análise [BRU 2010].

Os algoritmos de transformação de dados tem suas desvantagens. No caso da agregação com um operador de média, os dados transformados serão suavizados. Essa suavização pode ser prejudicial caso o comportamento original for demasiadamente heterogêneo, o que pode acontecer em aplicações paralelas com processos que efetuam processamentos diferentes. A suavização pode ser detectada através de técnicas de avaliação dos dados agregados [LAM 2013], permitindo o analista estar ciente dos problemas relacionadas à agregação. No caso de algoritmos de agrupamento, o maior problema é a seleção do processo mais representativo. Dependendo da métrica de similaridade utilizada e a quantidade de grupos definida, é possível que o processo mais representativo seja sempre muito longe do comportamento médio do grupo, gerando uma incoerência na análise. Técnicas de avaliação do agrupamento devem portanto ser aplicadas para detectar quando um grupo é muito diferente do comportamento representativo do mesmo.

1.4.4. Análise interativa por visualização de rastros

Técnicas de visualização para análise de desempenho estão presentes desde os primórdios da análise de aplicações paralelas e distribuídas. Ela consistem em transformar os rastros da aplicação paralela em representações visuais intuitivas que colocam em evidência padrões que podem eventualmente ser facilmente detectados pela interação do analista com a representação. A vantagem desta estratégia de análise é que ela pode contar com a interatividade e a experiência do analista durante o processo de visualização.

As técnicas de visualização de rastros podem ser divididas em três grupos de acordo com as características da representação: visualização comportamental, com os dados sendo desenhados ao longo de uma linha temporal; visualização estrutural, onde as entidades que são estudadas estão organizadas através de alguma topologia sem linha do tempo; e visualização estatística, agrupando todas as representações tradicionais como gráficos de dispersão.

Comportamental O exemplo mais conhecido e intuitivo de uma representação comportamental é o gráfico de espaço-tempo, derivado de gráficos de Gantt [WIL 2003]. Ela lista no eixo vertical todas as entidades observadas – processos, fluxos de execução, etc – algumas vezes organizadas hierarquicamente [KER 2000]. O comportamento de cada uma dessas entidades é desenhado horizontalmente utilizando retângulos para representar os estados da aplicação e flechas para representar as possíveis interações entre as entidades. Exemplos de ferramentas que fornecem este tipo de visualização são Vampir [BRU 2010], Paje [KER 2000] Projections [KAL 2006] e muitas outras [COU 2012, PIL 95, ZAK 99]. A grande vantagem de gráficos

espaço-tempo é a ênfase no aspecto temporal e na causalidade entre eventos, habilitando uma análise de desempenho de grão fino. Este tipo de representações são naturalmente limitadas pelo tamanho da tela. Somente algumas ferramentas, como Vampir, incorporam técnicas para evitar estes problemas de escalabilidade. A técnica de agrupamento (veja seção anterior) reduz a quantidade de entidades no eixo vertical através da supressão de processos cujo comportamento é muito semelhante [KNÜ 2007].

Estrutural Técnicas de visualização estruturais ilustram a estrutura da aplicação paralela ou do sistema computacional de execução. As técnicas que entram nesta classificação são as visualizações baseadas na topologia da rede, presentes em ferramenta tais como ParaGraph [HEA 91] e Viva [MEL 2013]; representações com três dimensões da ambientes em grade com a interconexão da rede [SCH 2009]; representações de grafo de chamadas como aparecem na ferramenta ParaProf [BEL 2003] e Virtue [SHA 99]; e matrizes de comunicação, implementadas em Vampir [BRU 2010] e outras ferramentas. A principal diferença de técnicas de visualização estruturais é que elas são independentes de uma linha do tempo; elas consideram os valores em um dado instante de tempo ou dados agregados temporalmente.

Estatística Técnicas de visualização estatística é a forma mais comum de representar dados. Elas consistem principalmente em gráficos estatísticos de dispersão, com duas ou mais variáveis correlacionadas. Vampir [BRU 2010] tem uma série de gráficos deste tipo, como gráficos de pizza [KER 2000], de funções [MIL 95], de processos. Diagramas de kiviat [HEA 91] e representações estatísticas em três dimensões [BEL 2003] são outros exemplos de representações.

A principal desvantagem das técnicas de visualização de rastros é a escalabilidade. Ela se manifesta tanto do lado do analista, único capaz de interagir com a representação dos rastros, tanto do lado técnico e das limitações das telas de computadores. As representações são concebidas de forma que o desenvolvedor ou o analista de desempenho seja capaz de perceber padrões. Neste contexto, é natural que quando a aplicação a ser analisada seja de larga escala, a quantidade de elementos a serem percorridos visualmente pelo analista se torna muito grande. Para se ter uma ideia de quão detalhista pode ser um rastro de aplicação paralela, basta considerar que um evento pode ser registrado sem muita intrusão a cada trinta microsegundos. Em um único segundo, cerca de trinta mil eventos poderiam ser registrados. Esperar que o analista possa visualmente percorrer todos estes eventos é irreal.

Por outro lado, as limitações das telas de computadores colocam uma barreira na quantidade de eventos que podem ser representados, portanto analisados, ao mesmo tempo. Considerando uma representação espaço-tempo e uma resolução horizontal de cerca de dois mil pixels, a maior quantidade de eventos que pode ser representada está na ordem de dois mil eventos. Nesta mesma representação, a maior quantidade de processos que podem aparecer ao mesmo tempo está na ordem de mil elementos (considerando uma resolução vertical de mil pixels). Levando em conta que as aplicações paralelas atuais podem chegar a dezenas de milhares de processos, uma análise por visualização de rastros limita o escopo de interpretações possíveis do comportamento.

A solução para o problema de escalabilidade das técnicas de visualização encontra-se na combinação desta com outras técnicas de análise. Algumas ferramentas [BRU 2010, MEL 2013] utilizam técnicas de transformação de dados como agregação e agrupamento

para permitir a análise de aplicações maiores. A agregação de dados espaço temporal pode ser benéfica para reduzir a complexidade e o detalhamento dos rastros de forma a obter uma visão mais geral do comportamento construída de forma coerente com o comportamento de cada processo que faz parte da aplicação. Esses dados mais gerais podem então ser representados visualmente através da mesma técnica de visualização para os dados originais, ou por outra técnica específica para os dados transformados. O mesmo ocorre com o uso de técnicas de agrupamento, onde algoritmos de redução de dados são utilizados para reduzir a quantidade de informação a ser representada.

Outra combinação possível para lidar com o problema da escalabilidade das técnicas de visualização é o uso de algoritmos de detecção automática de problemas de desempenho. Ainda que com suas limitações, como já discutido anteriormente, essa combinação pode ser benéfica pelo fato que algoritmos de mineração de dados podem ser utilizados para identificar os problemas conhecidos que, em seguida, são visualmente estudados pelo analista para tentar descobrir as razões do mau desempenho.

Qual a melhor técnica de análise?

A melhor técnica de análise depende diretamente do tipo de problema de desempenho que pode se manifestar no comportamento de uma aplicação paralela. Uma possível abordagem, considerando que não se conhece antecipadamente o comportamento da aplicação paralela, é começar com a análise de perfis de execução e em seguida utilizar abordagens mais sofisticadas, como análise automática e por visualização de rastros. Mesmo assim, é possível que a técnica ideal de análise não seja uma única, mas uma combinação de técnicas, cada qual levantando problemas de desempenho em níveis diferentes – globais ou locais – e permitindo ao analista ter uma compreensão melhor da sua aplicação paralela.

1.5. Ferramentas e bibliotecas

Ferramentas e bibliotecas tem por objetivo auxiliar o analista a realizar a análise de desempenho de aplicações paralelas. Atualmente, existem várias ferramentas que permitem uma fácil transição entre a teoria das técnicas apresentadas nas seções anteriores e a prática. Essas ferramentas e bibliotecas vão desde arcabouços que auxiliam a coleta e o registro do comportamento das aplicações até ferramentas sofisticadas de análise. Esta seção traz um subconjunto não exclusivo de ferramentas que podem ser utilizadas na análise de desempenho.

1.5.1. Coleta e registro

Score-P [VIH 2014] é uma infraestrutura de medição de desempenho altamente escalável que implementa as técnicas de amostragem e rastreamento. Ela integra vários formatos de arquivos capazes de registrar os resultados da amostragem e os rastros de execução. A grande vantagem de Score-P em relação a outras ferramentas da área é que ela oferece uma certa padronização, permitindo que os dados coletados através dela sejam analisados por uma série de ferramentas tais como Vampir [BRU 2010], Scalasca [GEI 2010], Periscope [BEN 2010], TAU [SHE 2006] e outros. Os formatos utilizados são o OTF2 [ESC 2011]

(*Open Trace Format 2*) para eventos de rastreamento, e o CUBE4 [GEI 2012] para perfis de execução.

TAU [SHE 2006] (*Tuning and Analysis Utilities*) é uma ferramenta portável que implementa, assim como Score-P, as técnicas de amostragem e rastreamento para programas paralelos escritos em várias linguagens tais como Fortran, C, C++, Java e Python. TAU realiza a coleta de informações através de técnicas avançadas de instrumentação que permitem registrar o comportamento de funções, métodos, blocos básicos, etc. Em suas últimas versões, TAU permite exportar rastros em formato compatível com Score-P, servindo então como uma outra ferramenta de análise.

EZTrace [TRA 2011] é uma ferramenta automática de rastreamento para aplicações de alto desempenho construídas com MPI e OpenMP. Ela utiliza o formato de rastro genérico Pajé [KER 2000, SCH 2014] e a primeira versão do formato de rastro OTF (*Open Trace Format*). O rastreamento implementado por EZTrace é baseado em pacotes e portanto extensível. O próprio analista pode definir novos pacotes para rastrear outros tipos de eventos na aplicação paralela sendo estudada. Outra facilidade da ferramenta é que o processo de interceptação de eventos é dinâmico, sem a necessidade de recompilação da aplicação a ser estudada.

Akypuera [SCH 2014a] consiste em um conjunto de ferramentas auxiliares em torno do formato de rastro Pajé [SCH 2014]. Além dessas ferramentas auxiliares, ela contém uma biblioteca de rastreamento para aplicações MPI que utiliza a librastro [SIL 2003]. Atualmente, existem conversores que traduzem conjuntos de arquivos de rastros dos formatos TAU, OTF e OTF2 para o formato Pajé.

SimGrid [CAS 2008] é um arcabouço para a construção de simuladores de sistemas paralelos e distribuídos. Ele pode ser utilizado como um instrumento para estudar o comportamento de sistemas como Grids, Clouds, HPC ou sistemas P2P. SimGrid tem embutido um módulo responsável pelo rastreamento das simulações, sendo capaz de gerar rastros em formato Pajé.

1.5.2. Análise

Vampir [BRU 2010] é uma ferramenta comercial de visualização de rastros altamente escalável que utiliza o formato OTF2 como base. Esta ferramenta se destaca por utilizar, opcionalmente, uma infraestrutura distribuída de computadores para ser capaz de representar em uma mesma tela o comportamento de dezenas de milhares de processos. Para tal, ela faz uso de técnicas de agrupamento e agregação de dados para diminuir a complexidade dos rastros antes da visualização.

Scalasca [GEI 2010] é uma ferramenta de análise de perfis de execução de aplicações paralelas altamente escalável. Ela implementa algoritmos de mineração de dados para análise automática dos rastros, listando as situações na qual a aplicação paralela demonstrou baixo desempenho. O analista pode investigar cada situação através de gráficos, com métricas adicionais que foram coletadas durante a execução da aplicação. Scalasca utiliza CUBE4 como formato de entrada, gerado pela infraestrutura Score-P.

Vite [COU 2009, COU 2012] é uma ferramenta de visualização de rastros que implementa a representação tradicional espaço-tempo utilizando OpenGL. Em virtude disto, ela é altamente escalável sendo capaz de representar uma grande quantidade de eventos. Vite utiliza Pajé e OTF como formato de entrada, e pode ser utilizada em combinação com EZTrace.

PajeNG [SCH 2014b] é uma ferramenta de visualização de rastros baseado em Pajé [KER 2000]. Atualmente, ela implementa apenas o simulador Pajé responsável pela criação do estado da aplicação paralela sob estudo a partir de seus rastros de execução. A vantagem desta ferramenta é a utilização do formato de rastro genérico Pajé [SCH 2014], capaz de adaptar a diferentes contextos semânticos de análise, tais como MPI, OpenMP, Java, GPU, entre outros.

Viva [SCH 2014c, MEL 2013] é uma ferramenta de visualização de rastros que implementa técnicas de visualização alternativas, tais como treemaps [SCH 2012a] e a visualização através de grafos hierárquicos [MEL 2013]. Viva utiliza PajeNG como simulador de rastros.

1.6. Conclusão

A análise de desempenho é uma etapa crucial e necessária durante o desenvolvimento de aplicações paralelas. Devido a magnitude dos sistemas paralelos atuais, a análise de desempenho é uma tarefa complexa a ser realizada, envolvendo inúmeros conceitos de sistemas computacionais.

Este capítulo apresentou os principais conceitos básicos a respeito da análise do desempenho de aplicações paralelas, desde efeito de sonda até o problema da sincronização de relógios. Em seguida, o capítulo trouxe um apanhado das técnicas de observação, coleta e registro do comportamento da aplicação, com evidência para as técnicas de baixa intrusividade como amostragem, cronometragem e contadores de *hardware* até uma técnica detalhista representada pelo rastreamento. Foram vistas em seguidas diferentes estratégias de análise de desempenho, da mais simples baseada em análise de perfis e automática até estratégias baseadas em transformação de dados e visualização de rastros. Por fim, foram vistas uma série de ferramentas atuais que podem ser utilizadas para se registrar e analisar o comportamento de aplicações paralelas.

1.7. Bibliografia

[AGU 2006] AGUILERA, G. et al. A systematic multi-step methodology for performance analysis of communication traces of distributed applications based on hierarchical clustering. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2006. IPDPS 2006. 20TH INTERNATIONAL, 2006. **Anais...** [S.l.: s.n.], 2006. p.8 pp.

[BEL 2003] BELL, R.; MALONY, A.; SHENDE, S. Paraprof: a portable, extensible, and scalable tool for parallel performance profile analysis. In: KOSCH, H.; BÖSZÖRMÉNYI, L.; HELLWAGNER, H. (Eds.). **Euro-par 2003 parallel processing**. [S.l.]: Springer Berlin / Heidelberg, 2003. p.17–26. (Lecture Notes in Computer Science, v.2790). 10.1007/978-3-540-45209-6_7.

[BEN 2010] BENEDICT, S.; PETKOV, V.; GERNDT, M. Periscope: an online-based distributed performance analysis tool. In: MÃ$\frac{1}{4}$LLER, M. S. et al. (Eds.).

Tools for high performance computing 2009. [S.l.]: Springer Berlin Heidelberg, 2010. p.1–16.

[BRO 2000] BROWNE, S. et al. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In: SUPERCOMPUTING, ACM/IEEE 2000 CONFERENCE, 2000. **Anais...** [S.l.: s.n.], 2000. p.42–42.

[BRU 2010] BRUNST, H. et al. Comprehensive performance tracking with vampir 7. In: MÃ$\frac{1}{4}$LLER, M. S. et al. (Eds.). **Tools for high performance computing 2009**. [S.l.]: Springer Berlin Heidelberg, 2010. p.17–29.

[CAS 2008] CASANOVA, H.; LEGRAND, A.; QUINSON, M. Simgrid: a generic framework for large-scale distributed experiments. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER MODELING AND SIMULATION, 10., 2008. **Anais...** [S.l.: s.n.], 2008.

[COU 2009] COULOMB, K. et al. **Visual trace explorer (vite)**. [S.l.]: October, 2009.

[COU 2012] COULOMB, K. et al. An Open-source Tool-chain for Performance Analysis. **Tools for High Perf. Computing 2011**, p.37–48, 2012.

[DAG 98] DAGUM, L.; MENON, R. Openmp: an industry-standard api for shared-memory programming. **IEEE Comput. Sci. Eng.**, Los Alamitos, CA, USA, v.5, n.1, p.46–55, 1998.

[DON 97] DONGARRA, J.; MEUER, H.; STROHMAIER, E. Top500 supercomputer sites. **Supercomputer**, v.13, p.89–111, 1997.

[DON 2013] DONGARRA, J. **Top500 list**.

[ESC 2011] ESCHWEILER, D. et al. Open trace format 2: the next generation of scalable trace formats and support libraries. In: PARCO, 2011. **Anais...** [S.l.: s.n.], 2011. p.481–490.

[GEI 2010] GEIMER, M. et al. The scalasca performance toolset architecture. **Concurrency and Computation: Practice and Experience**, v.22, n.6, p.702–719, 2010.

[GEI 2012] GEIMER, M. et al. Further improving the scalability of the scalasca toolset. In: INTERNATIONAL CONFERENCE ON APPLIED PARALLEL AND SCIENTIFIC COMPUTING - VOLUME 2, 10., 2012, Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2012. p.463–473. (PARA'10).

[GRO 94] GROPP, W.; LUSK, E.; SKJELLUM, A. **Using mpi**: portable parallel programming with the message-passing interface. Cambridge, MA, USA: MIT Press, 1994.

[HEA 91] HEATH, M.; ETHERIDGE, J. Visualizing the performance of parallel programs. **IEEE software**, v.8, n.5, p.29–39, 1991.

[JOS 2006] JOSHI, A. et al. Measuring benchmark similarity using inherent program characteristics. **IEEE Transactions on Computers**, Los Alamitos, CA, USA, v.55, p.769–782, 2006.

[KAL 2006] KALÉ, L. V. et al. Scaling applications to massively parallel machines using projections performance analysis tool. **Future Generation Comp. Syst.**, v.22, n.3, p.347–358, 2006.

[KER 2000] KERGOMMEAUX, J. C. de; OLIVEIRA STEIN, B. de; BERNARD, P. E. Pajé, an interactive visualization tool for tuning multi-threaded parallel applications. **Parallel Computing**, v.26, n.10, p.1253–1274, 2000.

[KNÜ 2007] KNÜPFER, A. et al. Visualization of repetitive patterns in event traces. In: APPLIED PARALLEL COMPUTING: STATE OF THE ART IN SCIENTIFIC COMPUTING, 8., 2007, Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2007. p.430–439. (PARA'06).

[LAM 2013] LAMARCHE-PERRIN, R. et al. **Evaluating Trace Aggregation Through Entropy Measures for Optimal Performance Visualization of Large Distributed Systems**. Grenoble, France: LIG, 2013. Research Report. (RR-LIG-037).

[LAM 78] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. **Communications of the ACM**, v.21, n.7, p.558–565, 1978.

[LEE 2008] LEE, C.; MENDES, C.; KALÉ, L. Towards scalable performance analysis and visualization through data reduction. In: IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING (IPDPS), 2008. **Anais...** [S.l.: s.n.], 2008. p.1–8.

[MEL 2013] MELLO SCHNORR, L.; LEGRAND, A.; VINCENT, J.-M. Interactive Analysis of Large Distributed Systems with Scalable Topology-based Visualization. In: INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS'13), 2013. **Anais...** IEEE Computer Society Press, 2013.

[MIL 95] MILLER, B. P. et al. The paradyn parallel performance measurement tool. **IEEE Computer**, v.28, n.11, p.37–46, 1995.

[PIL 95] PILLET, V. et al. Paraver: a tool to visualise and analyze parallel code. In: TRANSPUTER AND OCCAM DEVELOPMENTS, WOTUG-18., 1995, Amsterdam. **Proceedings...** [S.l.]: IOS Press, 1995. p.17–31. (Transputer and Occam Engineering, v.44).

[REE 94] REED, D. A. Experimental analysis of parallel systems: techniques and open problems. In: **Computer performance evaluation modelling techniques and tools**. [S.l.]: Springer, 1994. p.25–51.

[SCH 2009] SCHNORR, L. M.; HUARD, G.; NAVAUX, P. O. A. Visual mapping of program components to resources representation: a 3d analysis of grid

parallel applications. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 21., 2009. **Proceedings...** IEEE Computer Society, 2009.

[SCH 2012a] SCHNORR, L. M.; HUARD, G.; NAVAUX, P. O. A. A hierarchical aggregation model to achieve visualization scalability in the analysis of parallel applications. **Parallel Computing**, v.38, n.3, p.91 – 110, 2012.

[SCH 2012] SCHNORR, L. M.; LEGRAND, A.; VINCENT, J.-M. Detection and analysis of resource usage anomalies in large distributed systems through multi-scale visualization. **Concurrency and Computation: Practice and Experience**, v.24, n.15, p.1792–1816, 2012.

[SCH 2014] SCHNORR, L. M. **Pajé trace file format**. Porto Alegre, Brazil: UFRGS, 2014. `http://paje.sf.net`.

[SCH 2014a] SCHNORR, L. M. **Akypuera**. `http://github.com/schnorr/akypuera`.

[SCH 2014b] SCHNORR, L. M. **Pajeng – paje next generation**. `http://github.com/schnorr/pajeng`.

[SCH 2014c] SCHNORR, L. M. **Viva visualization tool**. `http://github.com/schnorr/pajeng`.

[SHA 99] SHAFFER, E. et al. Virtue: performance visualization of parallel and distributed applications. **Computer**, Los Alamitos, CA, USA, v.32, n.12, p.44–51, 1999.

[SHE 2006] SHENDE, S.; MALONY, A. The tau parallel performance system. **International Journal of High Performance Computing Applications**, v.20, n.2, p.287, 2006.

[SIL 2003] SILVA, G. J. da; SCHNORR, L. M.; STEIN, B. Jrastro: a trace agent for debugging multithreaded and distributed java programs. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 15., 2003. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003. p.46–54.

[STE 2001] STEIN, B. Depuração de programas paralelos. In: I ESCOLA REGIONAL DE ALTO DESEMPENHO DO ESTADO DO RIO GRANDE DO SUL, 2001, Gramado, RS. **Anais...** Sociedade Brasileira de Computação (SBC), 2001. p.151 – 176.

[TRA 2011] TRAHAY, F. et al. Eztrace: a generic framework for performance analysis. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2011 11TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2011. **Anais...** [S.l.: s.n.], 2011. p.618–619.

[VIH 2014] VI-HPS, V. I. H. P. S. **Score-p 1.x documentation**. [S.l.: s.n.], 2014.

[WIL 2003] WILSON, J. M. Gantt charts: a centenary appreciation. **European Journal of Operational Research**, v.149, n.2, p.430–437, September 2003.

[ZAK 99] ZAKI, O. et al. Toward scalable performance visualization with jumpshot. **International Journal of High Performance Computing Applications**, Thousand Oaks, CA, USA, v.13, n.3, p.277–288, 1999.