Monitoramento de Desempenho usando Dados de Proveniência e de Domínio durante a Execução de Aplicações Científicas

Renan Souza[1], Vítor Silva[1], Leonardo Neves[1], Daniel de Oliveira[2], Marta Mattoso[1]

[1]COPPE – Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brasil

[2]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

{renanfs,silva,marta}@cos.ufrj.br,lrmneves@poli.ufrj.br,
danielcmo@ic.uff.br

Resumo. *Simulações computacionais, em geral, são compostas pelo encadeamento de aplicações científicas e executadas em ambientes de processamento de alto desempenho. Tais execuções comumente apresentam gargalos associados ao fluxo de dados entre as aplicações. Diversas ferramentas de perfilagem de código têm apoiado a análise de dados de desempenho, como a Tuning and Analysis Utilities (TAU). Entretanto, essas ferramentas não favorecem as análises do fluxo de dados. Essas análises podem ser realizadas com a captura de dados de proveniência enriquecidos com dados de domínio extraídos ao longo da execução das simulações. Neste artigo, propomos o monitoramento de dados de desempenho por meio de consultas a uma base de dados de proveniência que integra dados sobre a execução, o fluxo de dados das simulações e os dados de domínio. Mostramos em aplicações científicas como consultas a essa base auxiliam no monitoramento de anomalias no desempenho, em tempo de execução.*

1. Introdução

Simulações computacionais de larga escala são comumente complexas e demandam paralelismo na execução em ambientes de Processamento de Alto Desempenho (PAD). Muitas dessas simulações envolvem a execução de diferentes aplicações, em que os dados gerados por uma aplicação são consumidos por outra, formando um fluxo de dados. Essas aplicações, em geral, manipulam arquivos que contêm um grande volume de dados, o que torna o processamento ainda mais complexo e demorado. Além disso, essas aplicações podem ser exploradas de diferentes formas dentro de uma simulação computacional. Normalmente, o cientista explora uma série de combinações possíveis dentre os fatores da simulação. Nessas explorações, a mesma simulação é executada diversas vezes utilizando diferentes combinações de parâmetros e métodos computacionais. Assim, minimizar essas combinações e detectar problemas de desempenho se torna uma prioridade para que o tempo total da simulação seja reduzido. Neste contexto, acompanhar o progresso da execução da simulação é fundamental para monitorar o desempenho e detectar anomalias em tempo de execução, tais como gargalos de comunicação entre processos e erros específicos da modelagem de parâmetros e dados do domínio de cada aplicação.

Para apoiar a análise de desempenho das aplicações em ambientes de PAD, diversas ferramentas de depuração e perfilagem de código têm sido utilizadas no suporte a simulações computacionais, como a ferramenta *Tuning and Analysis Utilities* (TAU) (Shende 2006). Com a TAU, é possível instrumentar o código das aplicações para capturar dados de desempenho e ainda realizar análises desses dados a partir de gráficos. Entretanto, são comuns situações em que o usuário precisa analisar o desempenho em conjunto com os dados do domínio das aplicações, além de estar ciente de todas as transformações nos dados ocorridas no fluxo. Por exemplo, em uma simulação de dinâmica dos fluidos computacional (Guerra *et al.* 2012), determinadas execuções geram erro ou demoram demasiadamente toda vez que o valor da tensão, calculado na simulação, é maior do que 250. Esse tipo de informação não é trivial de ser capturada e nem associada aos gráficos de desempenho, e somente com ela o usuário seria capaz de identificar problemas na sua simulação. Além da dificuldade de relacionar erros ou anomalias a dados específicos do domínio, tais ferramentas são focadas em fornecer dados de desempenho apenas após o término da execução (*i.e.* análise *post mortem*). Ademais, a TAU é intrusiva, uma vez que é necessário instrumentar o código da aplicação e nem sempre os usuários possuem tais códigos para realizar a adaptação. Logo, para monitorar a simulação em tempo de execução, utilizar estas ferramentas sem mecanismos adicionais dificulta (e até mesmo impossibilita) a análise em tempo de execução.

Uma forma de viabilizar a gerência e o monitoramento do desempenho em tempo de execução é modelar a simulação computacional como um *workflow* científico (Davidson e Freire 2008). Os *workflows* científicos têm sido usados para modelar, executar e monitorar simulações compostas pelo encadeamento de diversas aplicações, assim como o fluxo de dados envolvido. *Workflows* são executados por meio de Sistemas de Gerência de *Workflows* Científicos (SGWfC). Além de gerenciar a execução de *workflows,* os SGWfCs são responsáveis por capturar dados e metadados do estado da execução das aplicações que fazem parte do *workflow* ao longo do processamento. Esses dados podem ser armazenados em um banco de dados de proveniência para serem consultados sob demanda (Davidson e Freire 2008). Um banco de dados de proveniência armazena o histórico sobre a execução do *workflow*, desde informações sobre os programas envolvidos, dados consumidos e produzidos. Esse banco de dados de proveniência pode ser enriquecido com dados de desempenho de execução e dados de domínio da aplicação, permitindo análises mais detalhadas sobre a aplicação (Silva *et al.* 2014). Porém, os SGWfCs existentes não fornecem um arcabouço para depuração e perfilagem de código similar a ferramentas como a TAU. Acreditamos que se unirmos o poder de análise de ferramentas como a TAU ao banco de dados de proveniência, os usuários poderiam realizar as análises de desempenho levando em consideração o comportamento da geração dos dados de domínio das aplicações. Assim, se tais dados forem armazenados de forma estruturada em um Sistema de Gerência de Bancos de Dados (SGBD), serão passíveis de consulta e poderão servir de insumo para as análises de desempenho em ferramentas de perfilagem como a TAU.

Neste artigo, propomos o monitoramento de dados sobre o desempenho das aplicações que compõem o *workflow* em uma base de dados de proveniência e a sua integração aos dados de domínio nessa mesma base. Observa-se que a abordagem proposta não é intrusiva, pois não interferimos no código das aplicações que fazem parte do *workflow* gerenciado pelo SGWfC. Nos experimentos apresentados neste artigo,

utilizamos o SGWfC Chiron (Ogasawara *et al*. 2011) para gerenciar a execução do *workflow* em um ambiente de *cluster* e capturar os dados de proveniência, desempenho e domínio. O integramos à ferramenta TAU para prover visualizações para análise sobre o desempenho da simulação computacional.

O restante deste artigo está organizado como a seguir. Na Seção 2, apresentamos os trabalhos relacionados. Na Seção 3, mostramos o SGWfC Chiron. Na Seção 4, apresentamos a abordagem para o monitoramento e análise do desempenho da simulação em tempo real e a integração com a TAU. Na Seção 5, mostramos um estudo de caso com um *workflow* real e na Seção 6 concluímos este artigo.

2. Trabalhos Relacionados

A análise de desempenho e a depuração de erros em aplicações científicas paralelas ainda são grandes desafios para os desenvolvedores de sistemas. Diversas ferramentas instrumentam aplicações paralelas, tais como a TAU, o *Intel Parallel Studio XE* (Intel, 2015), e o *Unified Parallel C* (Almasi *et al.* 2011). TAU é composta por um conjunto de ferramentas para análise e depuração de programas paralelos desenvolvidos em Fortran, C, C++, UPC, Java e Python. A ferramenta possibilita a instrumentação e monitoramento de aplicações paralelas de forma intrusiva, permitindo o acesso a dados em diversos níveis, como uso de memória, I/O e desempenho do *hardware*. A TAU permite ainda a criação de visualizações tridimensionais de dados, propiciando uma análise gráfica do desempenho. Entretanto, a TAU não é capaz de associar os dados de desempenho aos dados específicos do domínio da aplicação, limitando assim a análise do comportamento da geração do fluxo de dados por parte do usuário.

O *Synthesized Tools for Archiving, Monitoring Performance and Enhanced DEbugging* (STAMPEDE) atua como uma aplicação intermediária que permite que usuários responsáveis por aplicações científicas complexas possam monitorar o estado de suas simulações computacionais em tempo de execução, detectar anomalias de execução automaticamente e corrigir problemas em tempo de execução (Gunter *et al.* 2011). O STAMPEDE conta com uma ferramenta de visualização gráfica de desempenho em tempo de execução. Contudo, o STAMPEDE não permite o uso de visualizações tridimensionais para análise de desempenho, considerando os dados de desempenho associados à computação e comunicação. Vale ressaltar que o suporte à coleta de dados de proveniência pelo STAMPEDE é realizado em outra base de dados.

Existem abordagens como o *Swift/T (*Wozniak *et al.* 2013*)* e o *AWARD (*Assuncao *et al.* 2012*)* que são capazes de representar e gerenciar o fluxo de dados entre as aplicações paralelas que compõem a simulação de forma escalável em ambientes de PAD. Soluções como essas facilitam a gerência de paralelismo de dados em simulações computacionais, mas não permitem uma análise de comportamento da execução que relacione os dados de domínio, de proveniência e de desempenho para depurar as simulações computacionais em tempo de execução. Dessa forma, destaca-se a ausência da capacidade de monitoramento durante a execução das simulações como a principal limitação de tais abordagens no contexto deste artigo.

3. O SGWfC Chiron

Para propiciar o monitoramento da simulação computacional em tempo de execução, nossa proposta é fundamentada em duas características que são oferecidas pelo SGWfC

Chiron: (i) uma álgebra de *workflows* centrada em dados, a qual permite a especificação do fluxo de dados, para apoiar a gerência do paralelismo das aplicações científicas e monitoramento do desempenho e (ii) a captura e o armazenamento estruturado em tempo de execução da proveniência dos dados associada ao fluxo de dados.

De acordo com Ogasawara *et al*., para especificar o *workflow* através da álgebra do Chiron, definimos quais serão as aplicações científicas gerenciadas e o encadeamento entre elas (2011). Além disso, é necessário especificar quais são os dados que devem ser consumidos e gerados; *i.e.*, como é o fluxo de dados a ser gerenciado com alto nível de detalhamento pelo SGWfC. Durante a execução do *workflow*, cada uma dessas aplicações é invocada diversas vezes, sendo que, para cada invocação, diferentes tuplas de entrada são consumidas e de saída são geradas. É essencial armazenar o estado de cada invocação distintamente para alcançar o alto nível de detalhamento sobre o perfil e estado da execução. Por essa razão, no Chiron, essas invocações são minuciosamente gerenciadas e as chamamos de *ativações*.

Em relação à arquitetura, no Chiron ela é organizada de forma distribuída em nós (*Chiron Nodes*, CN), que são máquinas onde o Chiron é instanciado e onde as aplicações científicas são efetivamente executadas. Todos os CN formam um *cluster* que é gerenciado de maneira "mestre-trabalhadores". O mestre é responsável por coordenar a distribuição das ativações, considerando as dependências de dados entre as aplicações que compõem a simulação. Além de gerenciar os trabalhadores, o mestre também é um trabalhador, *i.e.*, ele também executa ativações. Outra função importante do mestre é que ele é o único nó capaz de acessar a base de proveniência e realizar todas as operações de registro de dados para consulta. Todos os nós trabalhadores se conectam ao mestre por meio de uma rede de interconexão (*e.g., Infiniband*) e todos os CN acessam um sistema de arquivos compartilhado. O mestre também pode apresentar gargalos devido à centralização do controle em um único nó e ao fato de ele ser o único nó habilitado a acessar a base de proveniência. Contudo, o Chiron apresenta algoritmos de escalonamento que procuram minimizar o custo de comunicação entre nós, sendo essa considerada uma das principais causas de gargalo dessa arquitetura. Além disso, um servidor de banco de dados é instanciado com o SGBD centralizado a fim de reduzir também potenciais gargalos relacionados ao processamento de consultas no mestre.

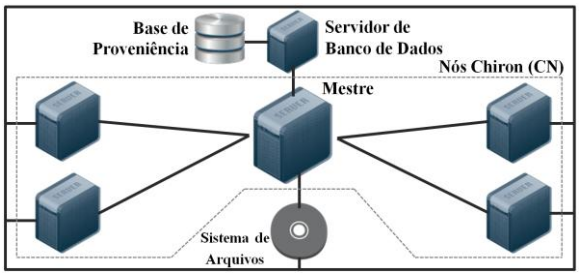

Figura 1. Arquitetura do Chiron

Quanto ao escalonamento das ativações, é utilizada uma interface de troca de mensagens, com a tecnologia MPI, para comunicação entre os CN. O nó mestre fica aguardando requisições dos trabalhadores. Um nó requisita trabalho e aguarda até que o mestre envie ativação. O mestre acessa a base de proveniência para recuperar as próximas ativações prontas para serem executadas e, em seguida, envia uma mensagem contendo uma ou mais ativações para o nó que requisitou, dependendo do modelo de execução selecionado pelo Chiron. O nó trabalhador recebe a ativação, executa-a e

envia uma mensagem de *feedback* para o mestre, contendo informações sobre a execução. Finalmente, o mestre recebe as informações enviadas pelo trabalhador, armazena a proveniência no banco de dados e fica aguardando por novas requisições. O término da execução é observado quando todas as ativações referentes a todas as aplicações do *workflow* completaram seu processamento. Semelhante ao armazenamento contínuo de dados de proveniência, é necessário também registrar dados sobre o desempenho da execução.

4. Monitoramento baseado em Proveniência e Dados de Domínio

A apresentação da abordagem proposta para realizar a análise de desempenho e o monitoramento é feita partindo das informações mais gerais sobre o desempenho chegando até detalhes mais específicos, relacionando dados de desempenho aos dados de domínio da aplicação. Ao fim, mostramos como realizamos uma integração de tais dados à ferramenta TAU, para prover análises visuais dos resultados.

4.1 Métricas Utilizadas

Sabe-se que métricas como *speedup* são extensivamente utilizadas para medir o desempenho de aplicações paralelas, inclusive de SGWfC paralelos. Elas são importantes para obter uma noção geral do ganho em desempenho da aplicação. Entretanto, para uma análise mais aprofundada do desempenho e para aumentar o entendimento dos resultados, métricas mais detalhadas são necessárias. Nossa proposta consiste em decompor o tempo total de execução em tempos de acordo com os potenciais gargalos relacionados à execução da simulação computacional (por exemplo, o gargalo de comunicação entre os recursos computacionais), capturá-los e armazená-los na base de proveniência para associá-los ao fluxo de dados e possibilitar consultas analíticas em tempo de execução.

Uma forma conhecida de detalhamento da avaliação é decompor o tempo total decorrido em tempos de computação útil (parte do tempo gasta com a aplicação sendo executada) e comunicação (parte do tempo gasta com comunicação entre processos) (Hennessy 2012). Assim, nossa proposta decompõe o tempo total decorrido da execução do *workflow* (T_{wf}) em tempo total de computação útil (T_{comp}) e o tempo de comunicação (T_{comm}). Além disso, para contemplar outro potencial gargalo, adicionamos o tempo total gasto acessando a base de proveniência (T_{prov}). Assim, $T_{wf} = T_{comp} + T_{comm} + T_{prov}$. Como visto na Seção 3, cada nó Chiron (CN) instancia o Chiron e executa as aplicações em paralelo. Logo, temos um $T_{wf}(i)$ para cada CN *i*, onde $i = [1, m]$ e m é o número de CN no *cluster*. Temos ainda que $T_{comp}(i)$ é o tempo total que o CN *i* gasta com computação útil, $T_{comm}(i)$ é o tempo total que o CN *i* gasta se comunicando com outros CN e $T_{prov}(i)$ é o tempo total que o CN *i* gastou acessando a base de proveniência. Logo, $T_{wf}(i) = T_{comp}(i) + T_{comm}(i) + T_{prov}(i)$. Como o tempo total decorrido em uma aplicação paralela é dado pelo CN que for mais lento, temos que $T_{wf} = \max\{T_{wf}(1), T_{wf}(2), \ldots, T_{wf}(m)\}$.

Podemos detalhar ainda mais o T_{comp}, pois no ambiente de PAD, cada CN pode ter múltiplos processadores e avaliar o desempenho distintamente de cada processador pode ser importante. Assim, definimos que $T_{comp}(i, j)$ é o tempo total que o processador *j* do CN *i* gastou com computação. Logo, podemos escrever que $T_{comp}(i) =$

$\sum_j T_{comp}(i,j)$. Destacamos que cada $T_{comp}(i,j)$ é armazenado sendo relacionado com a ativação do *workflow* que gastou esse tempo. Além disso, armazenamos no banco de proveniência outras informações específicas, como, por exemplo, os dados de entrada consumidos e os de saída gerados pela execução da ativação. Somando-se a isso, temos ainda registrada cada função de acesso à base de proveniência (*e.g.*, inserção de ativações e inserção das tuplas de saída geradas no decorrer do fluxo de dados). Por isso, o tempo total de proveniência $T_{prov}(i)$ visto pelo CN *i* é composto pela soma dos tempos de todas essas funções de proveniência realizadas pelo CN *i* durante a execução do *workflow*. Assim, definimos que $T_{prov}(i) = \sum_p T_{prov}(i,p)$, onde p é a função executada no banco de proveniência no SGBD. Como inserimos todos esses tempos definidos até então na base de proveniência, realçamos que é natural que uma das funções p seja a própria inserção desses dados de desempenho. Logo, por essa proposta, é possível também quantificar a sobrecarga adicionada para inserir tais dados sobre desempenho na base, como mostraremos na Seção 5. Finalmente, é importante observar que apenas o CN mestre executa funções na base de proveniência na arquitetura atual do Chiron. Consequentemente, supondo que $i = 1$ é o CN mestre, $\forall i \neq 1\ T_{prov}(i) = 0$.

4.2 Armazenamento da Proveniência

Todos os dados de proveniência capturados pelo Chiron são armazenados em uma base de dados relacional. Os dados capturados utilizando as métricas apresentadas na Seção 4.1 também precisam ser armazenados nessa base de dados. Para isso, é necessário uma modelagem de dados para a base de proveniência. Nesta Subseção, mostramos as principais tabelas da base de dados relacional do Chiron utilizadas para monitoramento da execução da simulação computacional.

4.2.1 Modelo de Proveniência do Chiron e Integração com Dados de Domínio

No Chiron, os dados de domínio são representados por meio de tabelas na modelagem da base de proveniência. Mais especificamente, utilizam-se referências nessas tabelas para ativações executadas para uma determinada simulação computacional. Por isso, é possível relacionar informações sobre o estado das execuções aos dados do domínio da aplicação. A Figura 2 ilustra um recorte do modelo de dados de proveniência atual do Chiron, apresentando apenas a parte da proveniência retrospectiva (*i.e.* dados de execução), foco deste artigo. Portanto, esse modelo de dados já considera as alterações que permitiram a integração com a TAU. Na tabela *EWorkflow* são armazenadas informações da execução corrente do *workflow* e na *EActivity* ficam informações de proveniência de cada atividade (associada a uma aplicação), como os tempos de início e de fim da execução da atividade. Na tabela *EFile* são armazenadas informações (*e.g.*, tamanho e diretório) dos arquivos manipulados na execução de cada ativação. Na *EMachine*, armazenamos detalhes sobre as máquinas do ambiente de PAD onde as ativações são executadas. A tabela *EActivation* é a mais importante para monitoramento do desempenho da execução do *workflow*. Nela, armazenamos informações importantes sobre o estado da execução de cada ativação, tais como se terminou com sucesso ou com falhas (*status* e *exitstatus*), quais erros foram lançados pelo programa (*terr*), qual a linha de comando utilizada na invocação (*commandline*), a quantidade de tentativas que falharam (*failure_tries*), e em qual nó (*machineid*) e processador (*processor*) a ativação está sendo executada. Assim, é possível monitorar o estado da execução com alto nível de detalhamento.

Um dos principais diferenciais de se utilizar dados de proveniência no Chiron é que existe a relação entre os dados de execução e os dados de domínio que correm pelo fluxo de dados. Na Figura 2, a tabela *InputFitPlane* é um exemplo de tabela de domínio com dados de entrada para uma determinada atividade. Nela, são armazenados todos os dados gerenciados pelo Chiron para tal atividade. Cada ativação da atividade consome uma tupla (ou um conjunto de tuplas, dependendo da atividade) na execução da ativação. Devido ao relacionamento com a tabela *EActivation*, é possível associar essa execução à tupla (ou ao conjunto de tuplas) da tabela do domínio. Dessa forma, podemos minuciosamente analisar, em tempo de execução, por meio de consultas SQL, qual dado específico do domínio levou a aplicação a, por exemplo, falhar várias vezes até conseguir executar com sucesso; ou, ainda, o dado que levou a aplicação a demorar mais do que a média; entre outras análises. Na Seção 5 mostramos exemplos de consulta que ilustram essa funcionalidade.

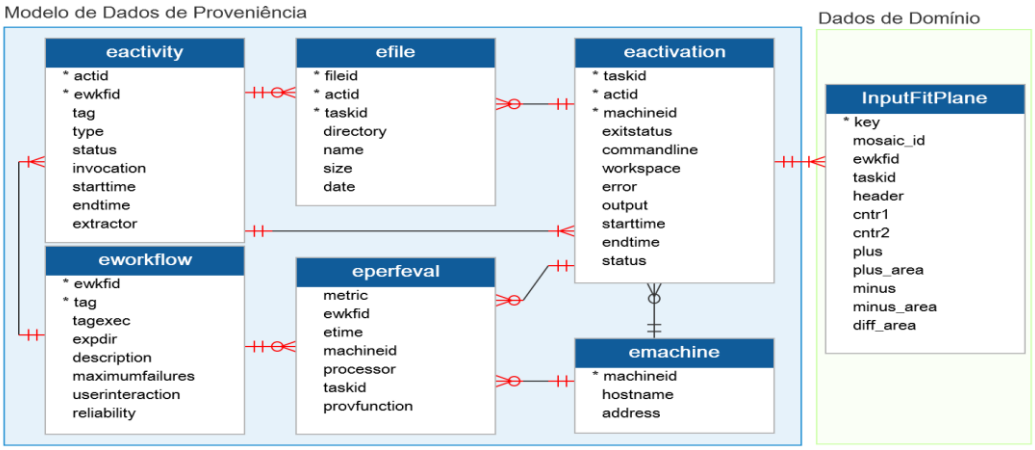

Figura 2. Recorte do modelo de dados do Chiron com foco na proveniência retrospectiva.

4.2.2 Tabela para dados sobre desempenho

Para armazenar todos os dados colhidos utilizando as métricas apresentadas na Subseção 4.1, foi necessário adicionar uma nova tabela ao banco de dados. Chamamos essa tabela de *EPerfEval* (Figura 2). De forma geral, é necessário identificar a execução do *workflow* associada a cada medida, o que é feito por meio do campo *ewkfid* (identificador de cada execução de *workflow* no banco de dados). Em seguida, armazenamos os tempos de cada uma das três métricas de forma diferenciada: (i) Para armazenar tempos da métrica T_{comp}, medimos o tempo decorrido (*etime*) para executar cada ativação. Relacionamos essa medida à ativação (por meio do campo *taskid* – identificador da ativação no banco de dados), à maquina (*machineid* – identificador do nó) e ao processador (*processor*) onde a ativação foi executada; (ii) para armazenar tempos de T_{comm}, ao enviar ou receber uma mensagem, contabilizamos o tempo (*etime*), identificamos a máquina (*machineid*) que está recebendo ou enviando mensagens ou ativações e armazenamos a ativação (*taskid*) associada à comunicação; (iii) para T_{prov}, o tempo decorrido (*etime*) de cada função (*provfunction*) do código do Chiron que acessa a base de proveniência (por meio de consultas SQL) é armazenado.

Como o propósito da tabela *EPerfEval* é executar consultas analíticas de forma eficiente, a mesma pode apresentar dados redundantes, pois o campo *processor* é replicado em outra tabela do banco de dados (*EActivation)*, cuja chave primária é o

campo *taskid*. Consequentemente, se seguirmos os princípios de banco de dados, esse campo não deveria estar na tabela *EPerfEval*. O objetivo dessa desnormalização é diminuir o número de junções com tabelas relacionadas, logo aumentamos o desempenho dessas consultas. Como os dados de desempenho são continuamente inseridos nessa tabela, ao longo da execução da simulação, a inserção precisa ser otimizada. É conhecido que índices em bancos de dados podem ser muito úteis e acelerar o desempenho de consultas. Contudo, a inserção em tabelas indexadas é mais custosa que em tabelas sem índices. Além disso, inserções com verificações de integridade referencial também são mais custosas, pois precisam analisar as chaves estrangeiras. Em contrapartida, já uma inserção simplificada sem nenhum índice ou restrição de integridade referencial tem baixo custo. Por essa razão, removemos todas as verificações de integridade em tempo de execução, uma vez que as inserções são bem controladas e com isso diminuímos consideravelmente o gargalo da captura dos dados de desempenho em tempo de execução.

Entretanto, é importante observar que apesar da inserção em tempo de execução ser otimizada e o impacto da captura dos dados de desempenho ser pequeno, seleções baseadas em predicados desta tabela são custosas. Isso é ainda mais crítico em simulações longas e que geram muitos dados de desempenho. Nos experimentos apresentados neste artigo, essa tabela possui mais de dez milhões de tuplas. Para resolver isso, adicionamos os devidos índices e restrições de integridade referenciais ao final da execução. Portanto, embora tais consultas sejam mais custosas em tempo de execução, é vantajoso adotar essas estratégias. Além de tornar possíveis as análises estruturadas do desempenho em tempo de execução, o tempo total da execução do *workflow* não é significativamente afetado. Ademais, as análises ao final da execução são otimizadas.

4.3 Integração com a TAU

Com intuito de se analisar, de forma mais abrangente, o desempenho das simulações paralelas, a integração com a TAU foi feita por meio da criação de um cartucho de conversão dos dados de desempenho (presentes na base de proveniência) para arquivos que contenham o perfil da execução capaz de permitir a visualização desses dados pela TAU. Esse cartucho é conhecido como *Parser-PerfEval* (baseado no termo em inglês *Parser for Performance Evaluation*, que significa conversor para avaliação de desempenho). Com o cartucho, TAU apoia a análise por diferentes pontos de vista utilizando-se dos dados armazenados no SGBD em tempo de execução. Por meio de parâmetros definidos pelo usuário, o *Parser-PerfEval* pode gerar resultados de comparação entre o tempo gasto por cada nó por três abordagens: (i) função desempenhada, que considera as operações de computação, comunicação e proveniência, (ii) métodos instrumentados no Chiron, que também considera as três operações de (i), e (iii) tempo de inatividade de cada nó referente a uma tarefa específica. Ainda, através dos parâmetros iniciais, pode-se controlar a unidade de tempo da análise, variando-se de milissegundos para *workflows* menores até semanas para *workflows* em maior escala, com a possibilidade de utilização de escala logarítmica.

Para isso, o *Parser-PerfEval* realiza o acesso à base de dados e obtém os dados de desempenho relevantes com os parâmetros definidos pelo usuário para a análise de desempenho. Ao acessar os dados, o *Parser-PerfEval* gera um arquivo no formato da TAU para expressar o perfil da execução, sendo esse arquivo conhecido como *profile.* Por exemplo, no caso de uma análise por função desempenhada, as entradas são

agrupada pelas operações de computação, comunicação e proveniência desempenhadas pelo Chiron. Com o uso do *Parser-PerfEval*, é possível analisar também o desempenho da simulação em vários níveis de granularidade. Verificando-se as ativações de acordo com seu tempo de início e tempo de término, podemos comparar o tempo real, dado pelo início e fim de sua ativação, com o tempo de execução de fato, determinando, assim, o tempo de inatividade de cada nó como a diferença entre eles. Conhecendo o tempo de inatividade do nó, por exemplo, podemos reajustar o balanceamento de carga por nó caso haja uma carga desproporcional em algum deles. Também é possível avaliar o balanceamento de carga por nó com relação ao tempo gasto para cada uma das funções desempenhadas. Por último, pode-se analisar o desempenho do próprio Chiron, destrinchando o tempo gasto por cada função de proveniência e comparando ao tempo de computação e comunicação em cada nó.

Os arquivos gerados são analisados graficamente pela ferramenta de visualização da própria TAU, por meio do comando *paraprof*. O formato compatível com essa ferramenta de perfilagem de código possibilita a criação de imagens como gráficos de barras, malhas tridimensionais e gráficos de dispersão, todos interativos e com a capacidade de customização inerente à ferramenta gráfica da TAU. Essa integração permite análises aprofundadas em gráficos gerados em poucos segundos e apoiam a tomada de decisões pelos cientistas em tempo de execução. A Figura 3 apresenta essas etapas para análise via ParaProf usando o *Parser-PerfEval*.

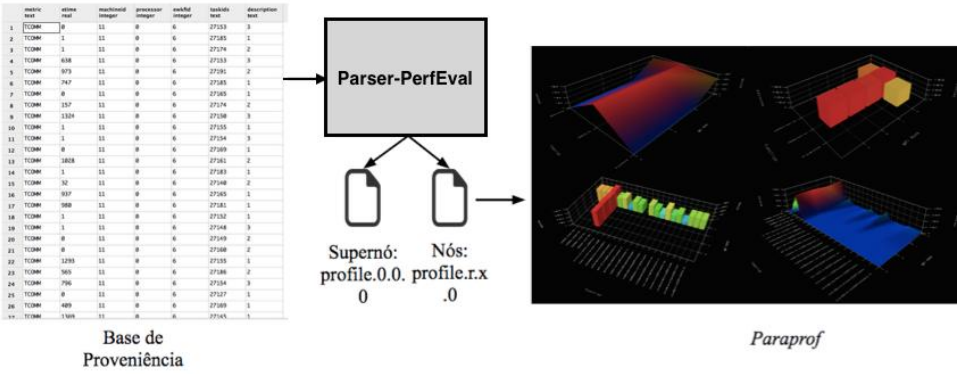

Figura 3 – Esquema de integração da base de proveniência com a TAU

5. Estudo de Caso

Esta seção apresenta um estudo de caso sobre um *workflow* real de astronomia, conhecido como *Montage* (Jacob *et al.* 2009), aplicando os conceitos da Seção 4. O Montage é um serviço para construção de mosaicos que auxilia cientistas nas análises astronômicas. É um *workflow* de larga escala que envolve a execução de diversas aplicações computacionalmente intensivas e que manipulam uma grande quantidade de dados em arquivos. Esse *workflow* é composto de nove aplicações científicas encadeadas. A especificação é ilustrada na Figura 4, na qual cada retângulo representa uma aplicação ou atividade do *workflow*, as setas representam o fluxo de dados e os círculos cheio e vazado representam início e fim do fluxo, respectivamente.

Executamos o *workflow* Montage no laboratório NACAD[1] da COPPE utilizando o ambiente SGI Altix ICE 8200 em uma arquitetura de memória distribuída, disco compartilhado, com quatro máquinas 2x Quad Core Intel Xeon X5355 2.66 GHz,

[1] Núcleo de Atendimento de Computação de Alto Desempenho – www.nacad.ufrj.br

totalizando 32 processadores. Como armazenamos os dados de desempenho no SGBD, podemos exemplificar o uso dos conceitos da Subseção 4.1 com consultas SQL. Por exemplo, a consulta da Figura 5 extrai o tempo de computação de cada máquina do *cluster* distintamente. Se o usuário ou um administrador do sistema utilizar essa consulta (ou qualquer outra de monitoramento) durante a execução do *workflow*, será possível monitorar o perfil da execução. A mesma consulta é capaz de extrair dados para as três métricas: tempo de computação (*TCOMP*) tempo de comunicação *(TCOMM*) e de proveniência (*TPROV*). Apresentamos os dados extraídos após a execução da consulta na Tabela 1 e na Figura 6. Além disso, utilizando os dados da Tabela 1 e as definições da Subseção 4.1, temos que o tempo total de execução T_{wf} do *workflow* é dado por $T_{wf}(3)$ = 1.010 min = 16,83 h.

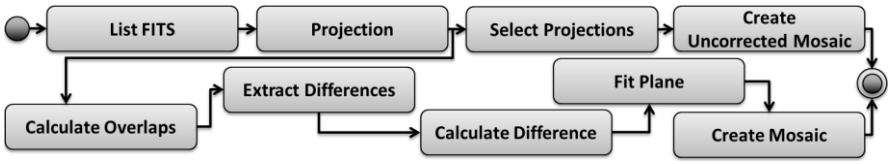

Figura 4 –*Workflow* Montage

```
SELECT metric, machineid, SUM(etime/(1000*60))
FROM EPerfEval WHERE metric IN ('TCOMP','TCOMM','TPROV')
GROUP BY machineid ORDER BY machineid;
```

Figura 5 – Consulta que mostra o tempo de computação para cada máquina do *cluster*

	Tcomp (min)	Tcomm (min)	Tprov (min)	Twf (min)
Nó (1)	533,33	96,33	24,00	675,00
Nó (2)	923,33	30,67	0	955,00
Nó (3)	993,33	33,00	0	1010,00
Nó (4)	961,67	32,67	0	943,33

Tabela 1 – Tcomp, Tcomm e Tprov para a execução do *workflow* Montage

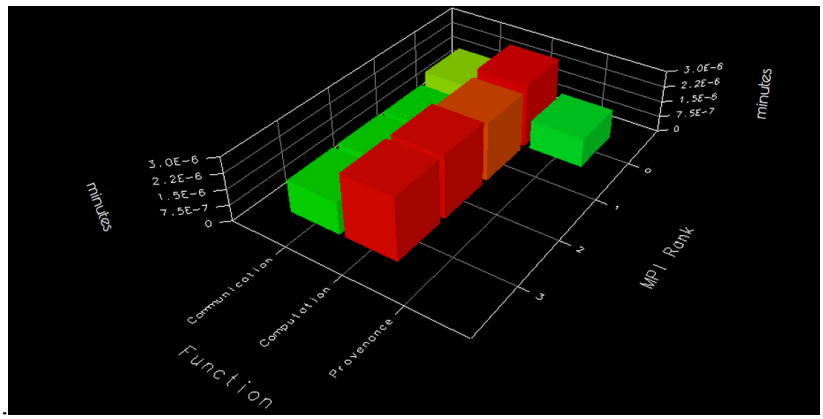

Figura 6 – Visualização tridimensional dos tempos gerais da execução gerada pela TAU, usando escala logarítmica.

Analisando a Tabela 1, pode-se identificar que o nó 1 é o CN mestre, pois é o único que apresenta $T_{prov} > 0$. Também observamos um desbalanceamento de carga computacional, pois o CN mestre executa menos computação que os outros 3 CNs. Além disso, há um gargalo de comunicação no CN mestre, *i.e.*, ele é o que apresenta maior carga de comunicação. Isso ocorre devido à centralização inerente à arquitetura

do Chiron. Adicionalmente, para prover uma melhor análise comparativa, utilizamos nossa integração com a ferramenta TAU para gerar um gráfico tridimensional usando escala logarítmica com base nos dados obtidos da tabela de dados de desempenho, como mostra a Figura 7. Na Figura 7, é possível verificar que, comparativamente, todas as máquinas apresentam tempo de computação balanceado. Os tempos de comunicação e de proveniência não são desprezíveis, apesar de apresentarem tempos com aproximadamente duas ordens de grandeza inferiores ao tempo de computação. É importante destacar que a ferramenta TAU disponibiliza recursos de *zoom*, de rotação e do tipo dos gráficos para facilitar a visualização no espaço tridimensional.

A próxima consulta, apresentada na Figura 8, é baseada na definição de $T_{comp}(i)$ e em dados de domínio do Montage. Mais especificamente, o usuário pode monitorar o desempenho das ativações executadas para a atividade de construir o mosaico com correções (*Create Mosaic*), limitando-se apenas uma região de interesse específica da imagem. Essa região da imagem é definida pelo conjunto de pixels em duas dimensões (representadas pelos atributos CRPIX1 e CRPIX2 que se encontram nas tabelas do banco de dados). Os resultados são ilustrados na Figura 9. A Figura 9 apresenta o balanceamento de carga do tempo de computação no processador de cada máquina e utilizamos escala logarítmica. Pode-se verificar que os processadores do nó 1, especialmente o processador 1, tem uma carga significativamente inferior aos processadores dos outros nós do *cluster* do Chiron, os quais apresentam um melhor balanceamento de carga nos processadores.

```
SELECT pe.machineid, pe.processor, SUM(pe.etime/(1000*60))
FROM EPerfEval pe, InputCreateMosaic cm
WHERE pe.metric='TCOMP' AND pe.taskid=cm.taskid
AND cm.crpix1 > 100 AND cm.crpix1 < 150
AND cm.crpix2 > 50 AND cm.crpix1 < 80
GROUP BY machineid, processor ORDER BY machineid, processor;
```

Figura 7 – Consulta para mostrar o balanceamento de carga no nível de processador

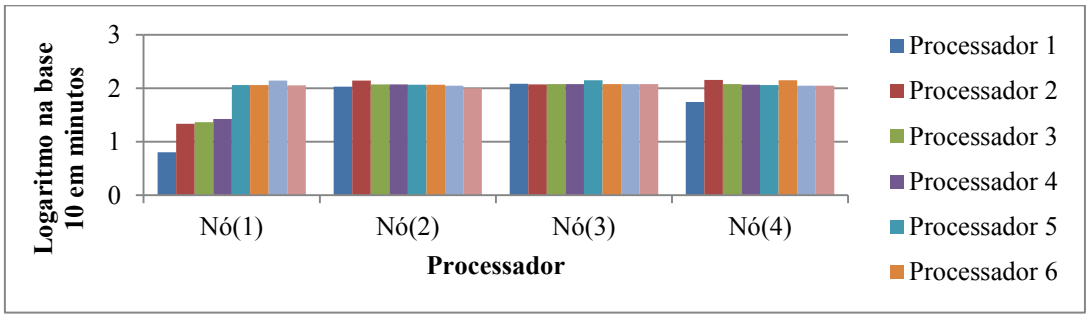

Figura 8 - Tempo de computação de cada processador de cada máquina

Já o gráfico da Figura 10 foi gerado baseado na definição de $T_{prov}(i)$. Na Figura 10, apresentamos apenas as funções de proveniência que apresentam tempo não desprezível, de modo a destacar as que possuem maiores gargalos. Dessa forma, além dos outros benefícios do monitoramento do desempenho a nível tão detalhado, é possível verificar anomalias. Identificamos que as funções *loadReadyActivation* (barra vermelha), *checkIfAllActivationsFinished* (barra amarela) e *loadActivation* (barra verde) são as que apresentam maiores gargalos atualmente. Esse resultado é particularmente importante para os desenvolvedores do SGWfC, pois esforços para possível otimização do sistema podem ser focados primeiramente nessas funções. Adicionalmente, podemos elaborar uma consulta sobre a tabela *EPerfEval* que avalie a sobrecarga de inserção dos

dados de desempenho nessa tabela, sendo assim possível quantificar o tempo adicional gerado pelo uso da abordagem proposta. Assim, verificamos que o tempo para inserir dados de desempenho na tabela foi de 6,73 min no total, o que corresponde a 0,6% do tempo total de execução. Assim, observamos que nossa proposta de monitoramento baseada na inserção e consulta de dados de desempenho não adiciona gargalos significativos no tempo total da execução do *workflow*.

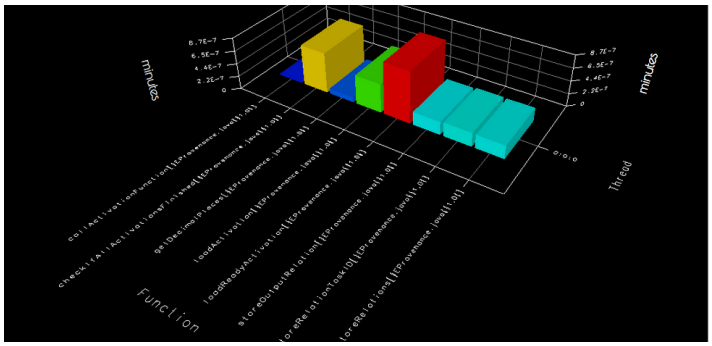

Figura 9 – Funções de proveniência que apresentaram maiores gargalos

```
SELECT processor, COUNT(*), AVG(etime/1000)
FROM EPerfEval WHERE machineid = 1
GROUP BY processor ORDER BY processor;
```

Figura 10 – Avaliação do balanceamento de carga dos processadores do nó 1

Processador	Número de ativações executadas	Tempo médio (s)
1	1	57,843
2	66	19,553
3	69	20,215
4	81	19,621
5	841	8,180
6	835	8,253
7	839	9,945
8	840	8,083

Tabela 2 – Resultados da consulta da Figura 4

Consultas analíticas também podem integrar dados do desempenho com dados de domínio do *workflow* Montage. Com o objetivo de entender o motivo de anomalias, suponhamos que, ainda durante a execução, foi identificado um comportamento incomum no perfil da execução. Por exemplo, o caso da Figura 9, em que observamos que o nó 1 executou notoriamente menos computação que os outros. Começamos a análise avaliando o tempo médio que cada processador do nó 1 levou executando as ativações e a quantidade de ativações executadas por cada processador, por meio da consulta da Figura 11. Nesse caso, observou-se (Tabela 2) que o tempo médio por ativação para o processador 1 é maior que o tempo médio dos outros processadores e que esse processador executou apenas 1 ativação enquanto que os outros executaram mais ativações. Abordagens análogas poderiam ser usadas para investigar por que os processadores 2-4 executaram menos que os processadores 5-8.

Devido ao relacionamento da tabela *EPerfEval* à tabela *EActivation*, pode-se facilmente realizar uma consulta para recuperar a única ativação executada pelo processador 1 e analisá-la mais a fundo. Com outras consultas sobre a tabela *EPerfEval*, observamos que os tempos de comunicação relacionados à tal ativação são desprezíveis. Nos dados de proveniência da ativação, podemos verificar que não houve nenhum erro

ou comportamento anômalo. Logo, precisaremos analisar os dados específicos do domínio para investigar o ocorrido. Para isso, utilizamos a referência à atividade à qual a ativação pertence e, assim, identificamos as tabelas do domínio que armazenam os dados de entrada e saída dessa atividade. A atividade em questão é a primeira do *workflow* Montage: *ListFITS*. Os dados específicos do domínio de astronomia são utilizados para criação de mosaicos. Em *ListFITS*, cada tupla de entrada gera várias tuplas que são consumidas a posteriori para criação do mosaico. Através de consultas SQL para analisar a tabela de saída dessa atividade, observamos que a cardinalidade do conjunto de saída referente à tupla de entrada consumida pela ativação anômala é consideravelmente maior em comparação com as outras ativações que executaram em menos tempo. Essa maior quantidade de tuplas fez com que a aplicação levasse mais tempo executando essa ativação, provocando o desbalanceamento de carga. Dessa maneira, através da nossa abordagem, foi possível identificar quais são os dados de entrada críticos, específicos do domínio da aplicação *ListFITS*. Verificamos que esses dados causam um gargalo na computação e provocam um desbalanceamento de carga na execução do *workflow* no Chiron.

Portanto, todas essas análises facilitam tanto o monitoramento do desempenho de forma geral, quanto as análises específicas que auxiliam a detecção de gargalos. Além disso, ainda possibilita a identificação de pontos críticos da máquina paralela de execução de *workflow*s que precisam cuidadosamente ser investigados. Enfatizamos, também, que quaisquer consultas apresentadas ou outras relacionadas ao desempenho podem ser executadas sobre o SGBD tanto em tempo de execução quanto ao final dela. Isso proporciona ao usuário, ou ao analista, formas flexíveis de se monitorar o desempenho de suas simulações computacionais, dado que o mesmo é capaz de observar anomalias durante o progresso da simulação e de realizar os tratamentos necessários, sem ter que esperar o término da execução dessa simulação.

6. Conclusões

Neste artigo propomos uma forma de monitorar e analisar dados de desempenho de simulações computacionais de larga escala por meio da integração dos mesmos a uma base de proveniência de um SGWfC. Apresentamos como nossa proposta pode facilitar o monitoramento das simulações em tempo real, através de consultas analíticas estruturadas sobre um SGBD. Mostramos que nossa abordagem algébrica permite análises detalhadas para detecção de anomalias associadas ao fluxo de dados, inclusive dados de domínio entre as aplicações científicas. Também integramos os dados de desempenho armazenados na base de proveniência à ferramenta de perfilagem de código TAU, de modo a proporcionar visualizações para apoiar análises. Finalmente, aplicamos nossa proposta em uma simulação computacional real do domínio de astronomia (i.e. *workflow* Montage). Dessa forma, acreditamos que nosso trabalho contribuiu para apresentar como é possível capturar e consultar dados de desempenho, integrando a dados de proveniência e a dados de domínio através de um SGWfC para gerenciar aplicações científicas em ambientes de PAD. Tais análises são impossíveis de serem realizadas sem o acoplamento com os dados de proveniência. Como trabalho futuro, pretende-se explorar outras métricas além das apresentadas neste artigo. Por exemplo, tempo de E/S gasto para manipular os arquivos, consumo de memória e de disco e o consumo de energia gasto pelos componentes do ambiente de PAD. Além disso, pretende-se desenvolver um mecanismo que execute automaticamente

determinadas consultas pré-definidas e que gere gráficos semelhantes aos apresentados automaticamente para facilitar o monitoramento detalhado da execução em tempo real.

7. Agradecimentos

Agradecemos à CAPES, à FAPERJ e ao CNPq por parcialmente financiar este trabalho. Agradecemos ao Centro de Computação Paralela da Intel no Brasil (*Intel Parallel Computing Center* – IPCC) localizado no NACAD/COPPE/UFRJ, que também apoia este trabalho.

8. Referências

Almasi, G., Hargrove, P., Tanase, G., Zheng, Y., (2011), "UPC Collectives Library 2.0". In: *Fifth Conference on Partitioned Global Address Space Programming Models (PGAS11)*

Assuncao, L., Goncalves, C., Cunha, J. C., (2012), "Autonomic Activities in the Execution of Scientific Workflows: Evaluation of the AWARD Framework". In: *Proceedings of 9th International Conference on Autonomic & Trusted Computing (UIC/ATC)Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*, p. 423 – 430, Fukuoka.

Davidson, S. B., Freire, J., (2008), "Provenance and Scientific Workflows: Challenges and Opportunities". In: *ACM SIGMOD International Conference on Management of Data*, p. 1345–1350, New York, NY, USA.

Guerra, G., Rochinha, F. A., Elias, R., de Oliveira, D., Ogasawara, E., Dias, J. F., Mattoso, M., Coutinho, A. L. G. A., (2012), "Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine", *International Journal for Uncertainty Quantification*, v. 2, n. 1, p. 53–71.

Gunter, D., Deelman, E., Samak, T., Brooks, C. H., Goode, M., Juve, G., Mehta, G., Moraes, P., Silva, F., et al., (2011), "Online workflow management and performance analysis with Stampede". In: *Proceedings of the 7th International Conference on Network and Service Management (CNSM)*, p. 1 –10

Hennessy, J. L., (2012), *Computer architecture: a quantitative approach*. 5th ed ed. Waltham, MA, Morgan Kaufmann/Elsevier.

Intel. Intel - Developer Zone. Disponível em: https://software.intel.com/en-us/intel-parallel-studio-xe/try-buy.

Jacob, J. C., Katz, D. S., Berriman, G. B., Good, J. C., Laity, A. C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., et al., (2009), "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking", *International Journal of Computational Science and Engineering (IJCSE)*, v. 4, n. 2, p. 73–87.

Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M., (2011), "An Algebraic Approach for Data-Centric Scientific Workflows", *Proceedings of the 37th International Conference on Very Large Data Bases (PVLDB)*, v. 4, n. 12, p. 1328–1339.

Shende, S. S., (2006), "The Tau Parallel Performance System", *International Journal of High Performance Computing Applications*, v. 20, n. 2 (May.), p. 287–311.

Wozniak, J. M., Armstrong, T. G., Wilde, M., Katz, D. S., Lusk, E., Foster, I. T., (2013), "Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing". In: *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, p. 95–102