



# UM MECANISMO DE TOLERÂNCIA A FALHAS EM EXECUÇÕES PARALELAS DE WORKFLOWS APOIADAS POR BANCO DE DADOS

Pedro Paiva Miranda

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheira.

Orientadores: Marta Lima de Queirós Mattoso

Renan Francisco Santos Souza

Rio de Janeiro

Agosto de 2015

UM MECANISMO DE TOLERÂNCIA A FALHAS EM EXECUÇÕES  
PARALELAS DE WORKFLOWS APOIADAS POR BANCO DE  
DADOS

Pedro Paiva Miranda

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO  
DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA  
POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO  
PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE  
ENGENHEIRA DE COMPUTAÇÃO E INFORMAÇÃO.

Examinada por:

---

Profa. Marta Lima de Queirós Mattoso, D.Sc.

---

Renan Francisco Santos Souza, B.Sc

---

Prof. Alexandre de Assis Bento Lima, D.Sc.

---

Vítor Silva Sousa, M.Sc.

RIO DE JANEIRO, RJ – BRASIL

AGOSTO de 2015

Miranda, Pedro Paiva

Um Mecanismo de Tolerância a Falhas em Execuções Paralelas de Workflows Apoiadas por Banco de Dados / Pedro Paiva Miranda. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2015.

X, 38 p.: il.; 29,7 cm.

Orientadores: Marta Lima de Queirós Mattoso e Renan Francisco Santos Souza

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2015.

Referências Bibliográficas: p. 27-28.

1. Tolerância a Falhas 2. Workflows Científicos 3. Proveniência de Dados I. Mattoso, Marta Lima de Queirós et al. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Computação e Informação. III. Título.

*À minha família e aos meus amigos.*

## AGRADECIMENTOS

Primeiramente, agradeço à minha família por sempre me apoiar, em especial ao meu pai e à minha mãe por me mostrarem quanto a educação é importante.

Agradeço a todos os professores que contribuíram para minha formação e em especial a professora Marta Mattoso por orientar este trabalho.

Agradeço ao Vítor Sousa e ao professor Alexandre de Assis Lima por participarem da banca.

Agradeço muito a Renan Souza pelo empenho em me orientar neste trabalho. E que neste curto período de tempo além de se tornar meu co-orientador e colega de trabalho, se tornou meu amigo.

Agradeço a meus amigos, em especial aqueles do colégio Franco-Brasileiro e aos companheiros ECI.

Agradeço aos colegas da IBM Research por me inspirarem.

Por fim, agradeço a todos os cidadãos brasileiros por investirem na UFRJ, que apesar de muitos problemas, entrega um ensino público de altíssima qualidade.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenharia de Computação e Informação.

## Um Mecanismo de Tolerância a Falhas em Execuções Paralelas de Workflows Apoiadas por Banco de Dados

Pedro Paiva Miranda

Agosto/2015

Orientadores: Marta Lima de Queirós Mattoso

Renan Francisco Santos Souza

Curso: Engenharia de Computação e Informação

Devido à alta demanda por poder de processamento que alguns *workflows* científicos possuem, muitos deles são gerenciados por Sistema de Gerência de *Workflows* Científicos (SGWfC) que apresentam técnicas de paralelismo em ambientes de alto desempenho com vários núcleos computacionais. Quanto maior o número de núcleos computacionais utilizados, maior a probabilidade de ocorrer falha em algum deles durante a execução do *workflow* científico. Nesse cenário é imprescindível que os SGWfCs implementem mecanismos de tolerância a falhas. Este trabalho tem como objetivo propor um mecanismo de tolerância a falhas em SGWfCs, que coletam dados de execução do *workflow* e os armazenam em bases de dados de proveniência em tempo de execução, e implementar esse mecanismo no SciCumulus/Cona Distributed Database System (d-SCC).

*Palavras-chave:* Tolerância a Falhas, *Workflows* Científicos, Proveniência de Dados.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

## A Fault Tolerance Mechanism for Parallel Execution of Workflows Relying on a Database

Pedro Paiva Miranda

August/2015

Advisors: Marta Lima de Queirós Mattoso

Renan Francisco Santos Souza

Major: Computer and Information Engineering

Due to the increasing demand for computing power required by scientific workflows, many of them are managed by Scientific Workflow Management Systems (SWfMSs) that use parallel techniques in high performance computing environments with a large number of computer nodes. As the number of used computer nodes grows, the probability of at least one of them fail during the scientific workflow running becomes higher. In this scenario, it is essential that SWfMSs implement fault tolerance mechanisms. This work proposes a fault tolerance mechanism in SGWfCs collecting workflow execution data and storing them in provenance databases at runtime, and implement this mechanism in SciCumulus/C on a Distributed Database System (d-SCC).

*Keywords:* Fault Tolerance, Scientific Workflows, Provenance Data.

## SUMÁRIO

1. Introdução.....	1
2. Fundamentação Teórica .....	4
2.1. SGWfCs e dados de proveniência .....	4
2.2. Falhas .....	5
2.3. Detectores de Falhas .....	6
2.4. Técnicas de Resiliência a Falhas.....	7
3. Mecanismo de Tolerância a Falhas em Execuções Paralelas de <i>Workflows</i> Apoiadas por Banco de Dados.....	9
3.1. Modelo do Sistema .....	9
3.2. Mecanismo Detector de Falhas .....	10
3.3. Mecanismo de Resiliência a Falhas .....	12
4. Implementação no SciCumulus/C on a Distributed Database System .....	14
4.1. SciCumulus/C on a Distributed Database System.....	14
4.2. Configuração de parâmetros .....	14
4.3. Implementação do Mecanismo Detector de Falhas .....	15
4.4. Implementação do Mecanismo de Resiliência a Falhas. ....	18
5. Análise Experimental .....	21
5.1. <i>Workflow</i> utilizado .....	22
5.2. Testes .....	22
6. Conclusão .....	25
6.1. Trabalhos Futuros .....	25
6.1.1. Testes de Escalabilidade.....	25
6.1.2. Supervisor Secundário.....	26
6.1.3. Escolha de parâmetros de forma dinâmica.....	26
Referências Bibliográficas.....	27



## LISTA DE FIGURAS

Figura 1. Esquema da tabela <i>emachine</i> . .....	16
Figura 2. Esquema da tabela <i>eactivation</i> . .....	18
Figura 3. <i>Workflow</i> utilizado nos testes.....	22
Figura 4. Resultados com tempos de execução .....	24

## LISTA DE SIGLAS

d-SCC – SciCumulus/Con a Distributed Database System

NC – Nó Computacional

PAD – Processamento de Alto Desempenho

SGBD – Sistema de Gerenciamento de Banco de Dados

SGBDD – Sistema de Gerenciamento de Banco de Dados Distribuído

SGWfC – Sistema de Gerência de *Workflows* Científicos

SQL – *Structured Query Language*

XML – *Extensible Markup Language*

## 1. Introdução

Experimentos científicos baseados em simulações computacionais de larga escala podem ser complexos e demandar paralelismo na execução em ambientes de Processamento de Alto Desempenho (PAD). Geralmente essas simulações são compostas por diferentes aplicações encadeadas, em que uma consome os dados gerados pela outra, formando um fluxo de dados. Essas simulações podem ser modeladas como *workflows* científicos, onde as atividades são formadas pelas aplicações e os relacionamentos são definidos pelo fluxo de dados, representando as dependências entre atividades em relação à produção e ao consumo dados (Deelman *et al.*, 2009). A execução de *workflows* pode ser gerenciada por Sistemas de Gerência de *Workflows* Científicos (SGWfC).

Os Sistemas de Gerência de *Workflows* Científicos devem armazenar dados de controle de execução com informações necessárias para gerenciar a execução paralela. Também devem guardar dados de proveniência que representam tanto a especificação do *workflow* quanto passos que geraram os resultados da execução e permitem a reprodutibilidade dos resultados, compartilhamento e reuso de conhecimento nas comunidades científicas (Davidson e Freire, 2008). Além dos dados de execução e proveniência, os SGWfCs devem armazenar dados de domínio que contêm informações específicas sobre o domínio da aplicação. Todos esses dados podem ser armazenados em tempo de execução em arquivos de *log* ou Sistemas de Gerenciamento de Banco de Dados (SGBDs). A vantagem de se armazenar os dados em SGBDs é que eles provem capacidade analítica em tempo de execução, permitindo descoberta antecipada de resultados (Oliveira *et al.*, 2014), monitoramento da execução associada ao dados de domínio gerados ao longo do fluxo de dados (Souza, Silva, Neves, Oliveira, & Mattoso, 2015) e execução interativa, conhecido como *user steering* (Mattoso *et al.*, 2015).

Além de armazenar tais dados, é imprescindível que os SGWfCs implementem mecanismos de tolerância a falhas em seus Nós Computacionais (NCs), ou seja, o sistema precisa ser capaz de completar a execução de *workflows* e gerar resultados corretos, mesmo que algum NC falhe. Essa necessidade se dá por causa do aumento de número de NCs usados em ambiente de PAD para realizar simulações cada vez maiores. Mesmo assumindo que a confiabilidade dos NCs e dos canais de comunicação entre eles é alta, à medida que o número de NCs aumenta para dezenas de milhares nesses ambientes de PAD, o período médio entre falhas deles diminui de alguns anos para algumas horas, ou menos (Chen *et al.*, 2005). Nesse cenário, em que o período médio entre falhas pode ser menor que o tempo de execução do *workflow*, é inviável utilizar SGWfCs que não sejam tolerantes a falhas dos NCs.

Entretanto, os SGWfCs existentes que armazenam dados de execução, de proveniência e de domínio usando SGBD em tempo de execução não proveem tolerância a falhas. Então apesar de possuírem vantagens relacionadas ao monitoramento da execução e análise dos dados em tempo de execução, não são alternativas viáveis para as execuções de *workflows* que utilizem muitos NCs.

Portanto, o objetivo deste trabalho é propor um mecanismo de tolerância a falhas para SGWfCs paralelos que armazenem dados de controle de execução, dados de proveniência e dados de domínio usando SGBD em tempo de execução. O uso do SGBD facilita o desenvolvimento de um mecanismo de tolerância a falhas, permitindo registrar e consultar quais NCs do sistema estão funcionando corretamente e quais falharam, redistribuir tarefas de NCs que falharam para NCs funcionando corretamente e armazenar os dados consumidos e gerados pelas tarefas que precisam ser recuperados caso alguma tarefa tenha que ser reexecutada.

Para implementar tal mecanismo proposto, foram adicionadas novas funcionalidades ao SciCumulus/Con a Distributed Database System (d-SCC) (Souza, 2015). O d-SCC é um SGWfC que armazena dados de proveniência em tempo de execução usando um Sistema de Gerenciamento de Banco de Dados Distribuído (SGBDD). Experimentos foram realizados para medir o impacto do mecanismo de tolerâncias a falhas no desempenho do sistema.

Esta monografia está organizada em seis capítulos. O Capítulo 2 aborda conceitos relevantes para o entendimento do tema. O Capítulo 3 apresenta o mecanismo de tolerância a falhas proposto pelo trabalho. No Capítulo 4 é mostrada a implementação do mecanismo proposto no d-SCC. A análise experimental realizada para verificar o desempenho da implementação é descrita no Capítulo 5. Por fim, o Capítulo 6 conclui o trabalho.

## 2. Fundamentação Teórica

Neste capítulo, são apresentados conceitos importantes para este trabalho. A Seção 2.1 apresenta os conceitos de *workflow* científico, Sistema de Gerência de *Workflows* Científicos e proveniência de dados. Na seção 2.2, são apresentados conceitos relacionados a falhas. Em seguida, na seção 2.3, é mostrada a definição de detectores de falhas. Por último, na seção 2.4, são abordadas as principais técnicas de resiliência a falhas.

### 2.1. SGWfCs e dados de proveniência

Sistemas de Gerência de *Workflows* Científicos (SGWfC) são responsáveis por controlar toda a execução de *workflows* científicos. Eles devem ser capazes de lidar eficientemente com o paralelismo para poder aproveitar da melhor maneira possível os recursos de ambientes de PAD. Além disso, os SGWfCs também devem armazenar 3 tipos de dados: (i) dados de controle de execução, (ii) dados de proveniência, e (iii) dados de domínio.

Dados de controle de execução englobam informações como que tarefas estão executando em qual processador, que dado de entrada determinada tarefa deve consumir etc. Dados de proveniência se referem às informações sobre o processo que gerou um dado e podem ser categorizadas como prospectiva e retrospectiva (Davidson & Freire, 2008). Proveniência prospectiva é a informação sobre a especificação da estrutura do *workflow*, enquanto a proveniência retrospectiva é informação sobre os passos que

geraram um determinado dado. Já os dados de domínio são os dados de entrada e saída das atividades e aqueles de principal interesse para os cientistas.

Esses três tipos de dados podem ser armazenados em banco de dados de proveniência, permitindo aos cientistas analisar a execução do *workflow* e até interagir com ela, caso os dados sejam armazenados em tempo de execução.

## 2.2. Falhas

Em um sistema distribuído, é necessário que cada componente desempenhe seu papel. No entanto, esses componentes podem falhar, ou seja, divergirem do comportamento esperado por quem os projetou. Se um sistema não for capaz de tolerar falhas, a falha de um de seus componentes pode fazer com que o sistema inteiro falhe. As maneiras como um componente pode falhar podem ser classificadas em diferentes tipos. Hadzilacos e Toueg (1994) listam alguns tipos de falhas:

- i. Falhas de processos:
  - a. *Crash*: Ocorre quando o processo encerra sua execução precipitadamente;
  - b. Omissão de Envio: Ocorre quando o processo deixa de enviar uma mensagem que deveria;
  - c. Omissão de Recebimento: Ocorre quando o processo deixa de receber uma mensagem destinada a ele;
  - d. Bizantina ou Arbitrária: Ocorre quando o processo exibe um comportamento arbitrariamente diferente do que deveria, como por exemplo, retornar um valor errado;

- ii. Falha por omissão de canais de comunicação: Ocorre quando o canal de comunicação não entrega uma mensagem que deveria.

As estratégias usadas por um sistema para tolerar falhas podem variar de acordo com os tipos de falhas que devem ser levados em conta. Nesse trabalho será considerado que um Nó Computacional (NC) falha se os seus canais de comunicação ou algum dos processos que ele está executando falharem. Caso contrário, o NC é considerado sem falhas.

### 2.3. Detectores de Falhas

Detectores de falhas são responsáveis por verificar quais recursos de um sistema distribuído não estão funcionando corretamente.

Duas abordagens são comumente utilizadas para detectar a ocorrência de falhas em recursos do sistema. A primeira delas, é chamada de modelo *push*, em que processos executando em cada NC, enviam periodicamente mensagens de pulsação (*heartbeat*) para o detector de falhas, anunciando que estão funcionando corretamente. Na ausência da mensagem de algum dos NCs durante um período pré-estabelecido, o detector de falhas considera que aquele NC falhou. Já na segunda abordagem, conhecida como modelo *pull* um processo no detector de falhas envia uma requisição periódica perguntando se os NCs estão funcionando corretamente. Na ausência de resposta de algum deles, o detector considera que aquele NC falhou (Garg e Singh, 2011).

Como em um sistema assíncrono não existe limite para o atraso de mensagens, não é possível fazer premissas baseadas no tempo. Assim, torna-se difícil determinar se um processo realmente falhou, se apenas está muito lento ou se as mensagens não chegaram (Coulouris *et al.*, 2005).



Para contornar essa situação, Chandra e Toueg (1996) propuseram o conceito de detector de falhas não confiável. Esse detector pode cometer erros, pois pode suspeitar que processos funcionando corretamente falharam ou, ainda, pode não suspeitar que um processo falhou, quando de fato falhou. Adicionalmente, os detectores de falhas não confiáveis podem ser caracterizados através de duas propriedades: completude e acurácia. A completude significa que existe um instante a partir do qual todos os processos que falharam são permanentemente considerados suspeitos por algum processo correto. Já a acurácia refere-se ao fato de existir um instante a partir do qual algum processo correto não é considerado suspeito por nenhum processo correto. Esses detectores, apesar de não serem confiáveis, têm utilidade na resolução de problemas em sistemas distribuídos assíncronos, em que existe a impossibilidade de implementação de um detector de falhas perfeito.

## 2.4. Técnicas de Resiliência a Falhas

As técnicas de resiliência são utilizadas por um sistema para se adaptar às falhas de alguns de seus componentes e permitir que ele continue executando corretamente. As duas principais técnicas de resiliência a falhas em sistemas que lidam com o paradigma de Processamento de Muitas Tarefas (MTC, do inglês *Many Task Computing*) (Raicu *et al.*, 2008) são *checkpoint-recovery* e replicação de tarefas (Garg e Singh, 2011), (Altameem, 2013).

A técnica de *checkpoint-recovery* consiste em salvar o estado do sistema em um dispositivo de armazenamento confiável, em determinados pontos da execução, *i.e.*, os *checkpoints*. Caso alguma falha ocorra durante a execução, o sistema é capaz de restaurar o último estado salvo e continuar a execução a partir desse ponto,

reexecutando apenas aquilo que foi feito após o *checkpoint*. Entretanto, é importante ressaltar que salvar o estado do sistema pode consumir recursos computacionais e, se for feito muito frequentemente, pode prejudicar o desempenho. Por outro lado, salvar muito espaçadamente pode significar uma grande perda de computação na presença de falhas (Garg e Singh, 2011). Por essa razão, é importante analisar as características do sistema relacionadas a falhas para a escolha ótima da frequência de *checkpoints*.

Além da técnica de *checkpoint-recovery*, a replicação de tarefas baseia-se na disposição de várias cópias da mesma tarefa em diferentes NCs. Dessa forma, se algum NC falhar, cópias da tarefa já estão sendo executadas em outro NC, evitando reexecuções dessas tarefas desde o começo. Ademais, ao usar essa técnica, a escolha do número de réplicas e de como é feita a distribuição delas são pontos fundamentais, já que ela exige que computações redundantes sejam feitas, diminuindo a eficiência do sistema (Garg e Singh, 2011).

### 3. Mecanismo de Tolerância a Falhas em Execuções Paralelas de *Workflows* Apoiadas por Banco de Dados

Neste capítulo, é apresentada a proposta para permitir tolerância a falhas em Nós Computacionais de Sistemas de Gerência de *Workflows* Científicos Paralelos que utilizam banco de dados de proveniência para auxiliar a gerência da execução. Para isso, é necessário que o sistema detecte a falha de um NC através do mecanismo detector de falhas (descrito na Seção 3.2). Após a detecção, é necessário utilizar um mecanismo de resiliência a falhas, descrito na seção 3.3, para que o sistema continue funcionando corretamente.

#### 3.1. Modelo do Sistema

A proposta do mecanismo de tolerância a falhas considera um SGWfC Paralelo e Assíncrono que distribua tarefas entre os Nós Computacionais, que serão responsáveis pela execução das mesmas. Também é considerado que esse sistema tenha uma base de proveniência confiável que seja atualizada em tempo de execução e que seja usada para apoiá-la.

Dos tipos de falhas descritos na seção 2.2, os seguintes são tolerados: *crash* e omissão de envio do processo, além de omissão do canal.

### 3.2. Mecanismo Detector de Falhas

O mecanismo detector de falhas proposto se baseia no modelo *push* (Garg e Singh, 2011) descrito na seção 2.3.

A proposta é que cada NC deve periodicamente calcular sua confiabilidade e atualizar esse valor no banco de dados. Um dos nós, chamado de Supervisor é responsável por consultar periodicamente o banco de dados e verificar se os NC estão com um valor de confiabilidade maior que um determinado limite e se esse valor foi atualizado dentro do período estipulado. Caso algum dos nós não atenda a um desses dois requisitos, o Supervisor considera que tal nó falhou.

Para apoiar esse mecanismo, o banco de dados foi modelado contendo uma relação para armazenar o estado de cada NC com os seguintes atributos:

**Tabela 1. Atributos do Estado do NCs**

Nome do Atributo	Descrição
node_id	Identificador do Nó Computacional
last_heartbeat	Horário da última atualização do valor de confiabilidade
reliability	Valor de confiabilidade
suspected_time	Horário em que o Nó Supervisor considerou que o NC falhou, ou nulo caso nunca tenha acontecido

A cada intervalo de tempo pré-determinado  $T$ , cada um dos Nós Computacionais  $NC_i$  atualiza a tupla referente a si no banco de dados, alterando os valores dos atributos *last\_heartbeat* com o horário atual e *reliability* com seu valor de confiança. A definição do valor de confiança a seguir, implementada no SciCumulus/C on a Distributed Database System (d-SCC) (Souza, 2015), foi utilizada nesta proposta.

$$reliability_i = \frac{processedTasks_i - failedTasks_i}{processedTasks_i}$$

onde  $processedTasks_i$  é o número total de tarefas processadas pelo  $NC_i$  e  $failedTasks_i$  número de tarefas processadas pelo  $NC_i$  que retornaram algum erro.

Com o propósito de identificar se algum NC falhou, a cada intervalo de tempo  $T + D$ , onde  $D$  é pré-determinado, o nó Supervisor consulta a relação com os estados dos NCs. Nessa consulta, ele verifica se existe alguma tupla, cujo valor do atributo *reliability* seja menor que um determinado limiar pré-determinado  $\theta$ , ou cuja diferença entre o horário atual e o valor do atributo *last\_heartbeat* seja maior que  $T + D$ . Caso alguma tupla atenda a essas condições, o NC a qual ela se refere é suspeito de ter falhado e o Supervisor atualiza os atributos *suspected\_time* com o horário atual e *reliability* com o valor 0. Após considerar que um NC falhou, o Supervisor aciona o mecanismo de resiliência a falhas, a ser descrito na próxima seção.

Quanto menor o valor de  $T$ , mais rápido o sistema identifica falhas. Contudo, a taxa de atualizações no banco de dados aumenta, podendo sobrecarregar os canais de comunicação e o SGBD. Essa sobrecarga pode afetar o desempenho do Sistema, principalmente quando o número de Nós Computacionais é muito grande, visto que o número de atualizações feitas a cada período  $T$  é igual ao número de NCs.

Em relação ao valor de  $D$ , esse deve ser o tempo gasto para que um NC registre sua última pulsação no banco de dados. Para a definição desse valor, idealmente, leva-se em conta os tempos de processamento dos NCs, dos SGBD e a latência do canal de comunicação. Entretanto, grande parte dos Sistemas Distribuídos é assíncrono e não existe um limite para definir esse tempo de atualização. Logo, uma estimativa deve ser feita. Se  $D$  for superestimado o sistema demorará mais para identificar uma falha. Em contrapartida, se  $D$  for subestimado, a acurácia do detector de falhas (Chandra e Toueg,

1996), diminuirá, já que NCs funcionando corretamente podem ser considerados suspeitos devido à demora na atualização de seu valor de confiança no banco de dados.

Já em relação a  $\theta$ , um valor muito baixo pode tornar o sistema pouco rigoroso, permitindo que NCs pouco confiáveis e que atrapalham o desempenho do sistema não sejam considerados falhados e que continuem executando e recebendo tarefas. Por outro lado, um valor muito alto para  $\theta$ , pode tornar o sistema muito rigoroso, considerando NCs que poderiam ajudar na execução de tarefas como falhados, comprometendo o desempenho do sistema.

Na proposta apresentada, a taxa de tarefas bem-sucedidas foi usada como único indicador para cada NC calcular seu valor de confiabilidade. Porém, outros indicadores podem ser usados para compor este valor.

### 3.3. Mecanismo de Resiliência a Falhas

O sistema modelado possui uma base de proveniência confiável atualizada em tempo de execução que pode ser convenientemente utilizada para salvar o estado do sistema em determinados pontos da execução, *i.e.*, os *checkpoints*. Por esse motivo, o mecanismo de resiliência a falhas proposto se baseia na técnica *checkpoint-recovery* (Garg e Singh, 2011) descrito na seção 2.4.

A proposta é que assim que o Supervisor considera que um NC falhou, ele se torna responsável por redistribuir automaticamente as tarefas que estavam atribuídas àquele NC que falhou para outros NCs que estejam funcionando corretamente e garantir que o NC falhado não receba novas tarefas. Para apoiar esse mecanismo de resiliência a falhas, o banco de dados foi modelado contendo uma relação com o estado de cada tarefa que possui os seguintes atributos:

**Tabela 2. Atributos do Estado das Tarefas**

Nome do Atributo	Descrição
task_id	Identificador da tarefa
node_id	Identificador do Nó Computacional ao qual a tarefa foi atribuída
status	Estado do progresso da execução.

Essa tabela é consultada pelos NCs para saber quais tarefas devem ser executadas por eles e o atributo *status* deve ser atualizado pelos mesmos, assim que houver alguma mudança no estado do progresso da execução. Quando algum NC for considerado como falhado, o Supervisor verifica quais tarefas devem ser redistribuídas, consultando no banco de dados quais tarefas não terminadas estavam atribuídas ao NC que falhou, através dos atributos *status* e *node\_id*. Em seguida, verifica qual NC funcionando corretamente possui a menor carga, ou seja, qual o NC tem menos tarefas não terminadas atribuídas a ele. Por fim, o Supervisor atualiza o valor de *node\_id* das tuplas referentes às tarefas que devem ser redistribuídas com o *node\_id* do NC com menor carga de trabalho (*i.e.*, número de tarefas).

## 4. Implementação no SciCumulus/C on a Distributed Database System

Neste Capítulo é exposta a implementação do mecanismo proposto no SGWfC SciCumulus/C on a Distributed Database System (d-SCC) (Souza, 2015).

### 4.1. SciCumulus/C on a Distributed Database System

SciCumulus/C on a Distributed Database System (d-SCC) (Souza, 2015) é um SGWfC paralelo baseado em álgebra de *workflows* científicos que armazena dados de proveniência em tempo execução em um Sistema de Gerência de Banco de Dados Distribuído (SGBDD). O d-SCC se encaixa no modelo de sistema descrito na seção 3.1. Seu SGBDD gerencia uma base de proveniência confiável, atualizada em tempo de execução e usada para apoiar a distribuição de tarefas.

Especificamente no d-SCC, um nó especial chamado de Nó Supervisor é responsável por gerar e distribuir ativações, que são a menor unidade de dados necessária para executar uma atividade (Özsu e Valduriez, 2011). As ativações são conhecidas como tarefas por outros sistemas (semelhante a subseção 3.2).

### 4.2. Configuração de parâmetros

O mecanismo detector de falhas proposto apresenta alguns parâmetros pré-determinados que podem influenciar no desempenho do sistema como discutido na seção 3.2. O d-SCC utiliza um arquivo de configuração XML (*Extensible Markup Language*) que contém a especificação conceitual e as propriedades da execução do



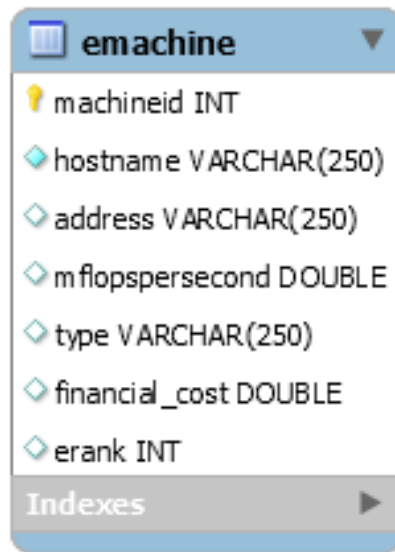
*workflow*. Neste trabalho esse arquivo de configuração foi estendido com um novo elemento para que o usuário pudesse definir os parâmetros ligados a tolerância a falhas. O elemento criado foi chamado de *faultTolerance* e possui os seguintes atributos que podem ser mapeados nos parâmetros utilizados na seção 3.2.

**Tabela 3. Atributos do elemento *faultTolerance***

Atributo	Parâmetro da seção 3.2	Descrição
active	Não se aplica	Liga ou desliga o mecanismo de tolerância a falhas
heartBeatInterval	$T$	Intervalo de tempo em segundos entre pulsações ( <i>heartbeats</i> )
hbLimitInterval	$T + D$	Intervalo de tempo em segundos em que um NC é considerado falhado, caso não envie nenhuma pulsação
limitReliability	$\theta$	Valor mínimo de confiança (seção 3.2) que um NC pode ter para não ser considerado falhado

#### 4.3. Implementação do Mecanismo Detector de Falhas

O d-SCC possui uma tabela com informações de cada Nó Computacional, chamada de *emachine*. Essa tabela foi aproveitada para apoiar o mecanismo detector de falhas e possui o seguinte esquema:



**Figura 1. Esquema da tabela *emachine*.**

Os atributos do mecanismo proposto na Seção 3.2 foram mapeadas nas seguintes colunas da tabela *emachine*, que foi estendida com novas colunas.

**Tabela 4. Mapeamento de Atributos na Tabela *emachine*.**

Nome do Atributo	Colunas da tabela <i>emachine</i>
node_id	machineid
last_heartbeat	Coluna last_heartbeat adicionada
reliability	Coluna reliability adicionada
suspected_time	Coluna suspected_time adicionada

Cada um dos NCs executa uma *thread* que, periodicamente, calcula sua confiança e a atualiza no banco de dados através da seguinte consulta SQL (*Structured Query Language*).

```
UPDATE emachine SET

    last_heartbeat = NOW() ,

    reliability = nc_reliability

WHERE machineid = node_id;
```

Onde `nc_reliability` é o valor de confiança, calculado usando a equação apresentada na seção 3.2, e `node_id` é o identificador do NC.

O nó Supervisor, além de atualizar a confiança, verifica se alguma máquina, que ainda não foi marcada, deve ser marcada como suspeita. Isso é feito utilizando a consulta SQL a seguir:

```
SELECT  * FROM emachine

WHERE last_heartbeat < (NOW() - INTERVAL hbLimitInterval SECOND)

AND suspected_time IS NULL;
```

Onde `hbLimitInterval` é o tempo limite para um nó ser considerado falhado como especificado na seção 4.2. Caso a consulta anterior retorne algum NC, ele deve ser marcado como suspeito de ter falhado, através da consulta:

```
UPDATE emachine SET

    suspected_time = NOW() ,

    reliability = 0

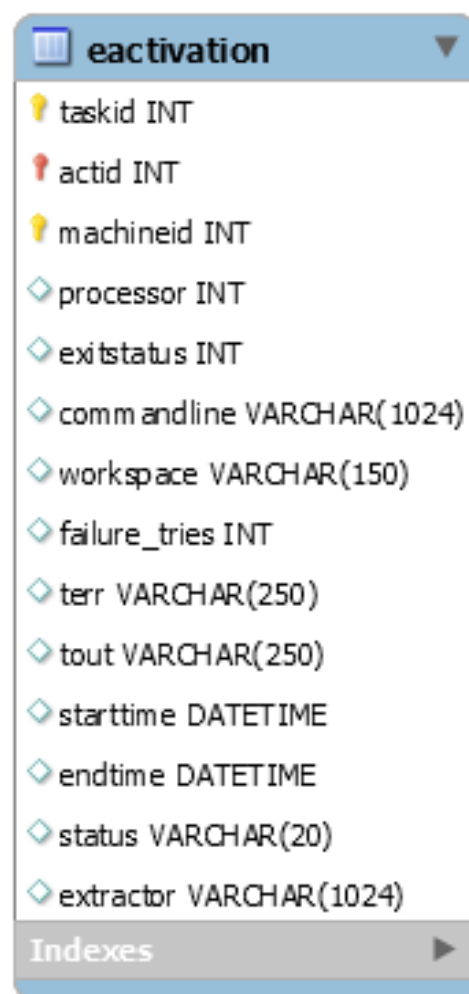
WHERE machineid = failed_node_id;
```

Onde `failed_node_id` é o id do NC que falhou. Em seguida, o nó Supervisor deve iniciar o mecanismo de resiliência a falhas descrito na próxima seção.

#### 4.4. Implementação do Mecanismo de Resiliência a Falhas.

Quando algum NC é marcado como suspeito de ter falhado, o nó Supervisor deve redistribuir as tarefas não concluídas atribuídas àquele NC que falhou para outro NC que esteja funcionando corretamente.

O d-SCC tem uma tabela contendo o estado de cada ativação chamada *eactivation* com seguinte esquema:



eactivation	
taskid	INT
actid	INT
machineid	INT
processor	INT
exitstatus	INT
commandline	VARCHAR(1024)
workspace	VARCHAR(150)
failure_tries	INT
terr	VARCHAR(250)
tout	VARCHAR(250)
starttime	DATETIME
endtime	DATETIME
status	VARCHAR(20)
extractor	VARCHAR(1024)
Indexes	

**Figura 2.** Esquema da tabela *eactivation*.

As colunas dessa tabela foram mapeadas nos atributos descritos na seção 3.3, com o objetivo de dar suporte ao mecanismo de resiliência a falhas.

**Tabela 5 Mapeamento de Atributos na Tabela *eactivation*.**

Nome do Atributo	Colunas da tabela <i>eactivation</i>
task_id	taskid
node_id	machineid
status	status

Para redistribuir todas as ativações não finalizadas de algum NC para outro, o supervisor faz a seguinte consulta:

```
UPDATE eactivation SET

    machineid = new_node_id,

    status = CASE

                WHEN status='RUNNING' THEN 'READY'

                ELSE status

            END,

    processor = NULL

WHERE machineid = failed_node_id

AND status !=\ 'FINISHED\ ';
```

Onde `failed_node_id` é o identificador do NC falhado e `new_node_id` o identificador do NC escolhido para receber as tarefas. Nesse caso, considera-se que o NC escolhido para receber as tarefas do NC com falha(s) deve apresentar a menor carga de tarefas não terminadas e que esteja funcionando corretamente. Escolha esse feita com a seguinte consulta:

```
SELECT m.machineid
FROM emachine m
LEFT JOIN eactivation a ON m.machineid = a.machineid
        AND a.status != 'FINISHED'
WHERE m.suspected_time IS NULL
GROUP BY a.machineid
ORDER BY COUNT(a.taskid)
LIMIT 1;
```

O nó Supervisor também é responsável por impedir que o nó que falhou receba novas tarefas. Como no d-SCC, o próprio nó Supervisor é quem distribui as ativações, basta ele remover o NC que falhou da sua lista de nós que podem receber novas ativações.

## 5. Análise Experimental

Neste capítulo é feita a análise experimental da implementação do mecanismo de tolerância a falhas proposto neste trabalho. Foram feitos testes comparativos entre o desempenho da execução com e sem o mecanismo de tolerância a falhas, além de validações para mostrar que a solução é capaz de tolerar falhas de seus Nós Computacionais (NC). Todos os testes foram executados no cluster Uranus, pertencente ao Núcleo Avançado de Computação de Alto Desempenho (NACAD) da COPPE/UFRJ. A Uranus é uma máquina SGI Altix ICE 8400 com as seguintes especificações retiradas de (NACAD, 2015):

128 CPUs Intel Xeon: 640 Cores

- 64 CPUs Six Core Intel Xeon X5650 (Westmere), 2.67 GHz: 384 Cores
- 64 CPUs Quad Core Intel Xeon X5355 (Clovertown), 2.66 GHz: 256 Cores
- Nós de processamento: 64
- Memória: 1.28 TBytes RAM (distribuída)
- Armazenamento em disco: SGI InfiniteStorage NAS (72 TBytes)
- Rede: Infiniband QDR, DDR e Gigabit
- Sistema operacional: Suse Linux Enterprise Server (SLES) + SGI Performance Suite
- Compiladores: Intel e GNU (Fortran-90 e C/C++) com suporte OpenMP
- MPI: MPT (SGI® Message Passing Toolkit), MVAPICH2 e OpenMPI

### 5.1. *Workflow* utilizado

Todos os testes foram realizados usando o *Scientific Workflow Benchmark* (SWB) (Chirigati *et al.*, 2012). No SWB, *workflows* sintéticos centrados em dados podem ser gerados, estipulando-se o tamanho do problema e a complexidade de cada computação.

Nesse trabalho foi utilizado um *workflow* SWB com 3 atividades regidas pelo operador algébrico *Map* com dependências entre elas como mostrado na figura a seguir.



**Figura 3. *Workflow* utilizado nos testes**

Nos testes executados o tamanho do problema estipulado foi de 1000 tarefas por atividade, totalizando 3.000 tarefas e com a complexidade de cada computação de 8 segundos em média.

### 5.2. Testes

Foram executados quatro testes e em todos eles foram utilizados 4 nós da Uranus, cada um com 8 cores. Nos testes em que o mecanismo de tolerâncias a falhas estava ligado os parâmetros especificados no arquivo de configuração do *workflow* foram:

```
<faultTolerance active="true" heartBeatInterval="35"  
hbLimitInterval="45" limitReliability="0.5"/>
```

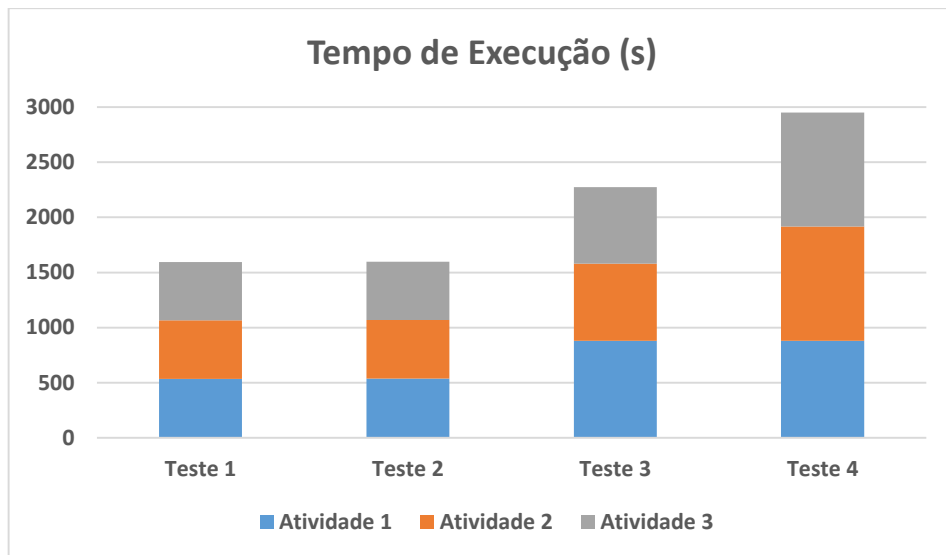
Para provocar a falha de um NC o comando a seguir é enviado para os nós alvos através de *Remote Shell* (*rsh*).



```
killall -9 java
```

Dessa forma, o d-SCC que executa na Java Virtual Machine (JVM) é terminado, provocando uma falha do tipo *crash* (seção 2.2). No Teste 1, o *workflow* foi executado com o mecanismo de tolerâncias a falhas desligado e nenhuma falha foi provocada. No Teste 2, o *workflow* foi executado com o mecanismo de tolerâncias a falhas ligado e nenhuma falha foi provocada. No Teste 3, o *workflow* foi executado com o mecanismo de tolerâncias a falhas ligado e a falha de 1 NC foi provocada 3 minutos após o início da execução do *workflow*. No Teste 4, o *workflow* foi executado com o mecanismo de falhas ligado e a falha de 2 NCs foi provocada, a primeira 3 minutos e a segunda 10 minutos após o início da execução do *workflow*. Vale ressaltar que o nó Supervisor não foi escolhido para falhar já que na implementação desse trabalho ele é o único responsável por distribuir as tarefas e consiste em um ponto único de falha do sistema.

Todos os testes foram executados 2 vezes e foram considerado nos resultados as médias dos tempos de execução das atividades de cada um dos testes. Os resultados são mostrados na Figura 4.



**Figura 4. Resultados com tempos de execução**

Pelos resultados dos tempos de execução, podemos perceber que para a escala de NCs utilizados no teste, o impacto no desempenho causado pelo mecanismo de tolerância a falhas é imperceptível, já que o tempo total de execução do Teste 1 foi de 1593s contra 1597s do Teste 2. Seria necessário fazer um teste em um ambiente com mais NCs para testar a escalabilidade do mecanismo. O Teste 3 e o Teste 4 validaram que o mecanismo funciona, já que mesmo com até metade dos NCs parando de funcionar o SGWfC conseguiu completar a execução corretamente, enquanto que na versão original do d-SCC a falha de um único NC inviabilizaria o término correto da execução.

Nos testes que provocaram falhas também foi medido o tempo que o nó Supervisor levou para detectar que um dos NCs tinha falhado. Este tempo foi em média de 70s.

## 6. Conclusão

Em um cenário que os cientistas precisam executar experimentos cada vez maiores e que demandam a utilização de quantidades muito grandes de NCs em um ambiente de PAD, se torna muito provável que algum desses NCs venha a falhar durante a execução do experimento. (Chen *et al.*2005). Por essa razão, Sistemas de Gerência de *Workflows* Científicos que apoiam esse tipo de experimento necessitam tolerar falhas, ou seja continuar a execução do *workflow* mesmo que um dos NCs falhe.

Visando a atender a esta necessidade, esse trabalho propôs um mecanismo de tolerância a falhas para SGWfCs paralelos que armazenem dados de controle de execução, dados de proveniência e dados de domínio usando SGBD em tempo de execução. O uso de SGBD facilita o desenvolvimento de mecanismos de tolerância a falhas já que eles disponibilizam informações valiosas sobre o estado da execução e permitem redistribuir as tarefas de maneiras simples, apenas fazendo atualizações no banco de dados. Além disso, permite que o usuário verifique quais NCs estão funcionando corretamente e interaja com a execução.

Este mecanismo proposto foi implementado, adicionando funcionalidades ao d-SCC (Souza, 2015) e testado. Os testes demonstraram que o mecanismo funciona, permitindo que *workflows* terminem suas execuções até mesmo com metade de seus NCs falhados.

### 6.1. Trabalhos Futuros

#### 6.1.1. Testes de Escalabilidade

Os testes realizados com a implementação contemplaram apenas uma escala muito pequena em ambientes de PAD. Por isso, ainda seria relevante avaliar o desempenho da solução em ambiente com mais NCs.

#### 6.1.2. Supervisor Secundário

Na proposta apresentada, o nó Supervisor é o único responsável por identificar se algum NC falhou e acionar o mecanismo de resiliência a falhas. Para evitar que ele se torne um ponto único de falha para o sistema, é necessário que um segundo nó, chamado de Supervisor Secundário, verifique periodicamente se o nó Supervisor está funcionando corretamente; caso contrário, o Supervisor Secundário deve assumir o papel do Supervisor.

#### 6.1.3. Escolha de parâmetros de forma dinâmica

Os parâmetros listados na seção 4.2 deste trabalho devem ser configurados pelo usuário antes da execução e caso sejam escolhidos de maneira ruim, podem atrapalhar o desempenho desse sistema. O cenário ideal consistiria no ajuste dinâmico desses parâmetros durante a execução do *workflow* para valores ótimos de acordo com as informações presentes no banco de dados de proveniência.

## Referências Bibliográficas

- Altameem, T., (2013). "Fault Tolerance Techniques in Grid Computing Systems". *International Journal of Computer Science and Information Technologies*, 4(6), pp. 858-862.
- Chandra, T. D., Toueg, S., (1996). "Unreliable failure detectors for reliable distributed systems". *J. ACM* 43.
- Chen, Z., Fagg, G. E., Gabriel, E., et al., (2005). "Fault tolerant high performance computing by a coding approach". *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming* (pp. 213-223). ACM.
- Chirigati, F., Silva, V., Ogasawara, E., et al., (2012). "Evaluating Parameter Sweep Workflows in High Performance Computing". *1st International Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET'12) SIGMOD/PODS 2012*, (p. 10). Scottsdale, AZ, EUA.
- Coulouris, G., Dollimore, J., Kindberg, T., (2005). "Distributed Systems: Concepts and Design (4 ed.)". Addison-Wesley Publishing Company.
- Davidson, S. B., Freire, J., (2008). "Provenance and Scientific Workflows: Challenges and Opportunities". *Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08)*. ACM, New York, NY, EUA.
- Deelman, E., Gannon, D., Shields, M., et al., (2009). "Workflows and e-Science: An overview of workflow system features and capabilities". *Future Generation Computer Systems*, 25(25), pp. 528–540.
- Garg, R., Singh, A. K., (2011). "Fault tolerance in grid computing: state of the art and open issues". *International Journal of Computer Science & Engineering Survey (IJCSES)*, 2(1), pp. 88-97.
- Hadzilacos, V., Toueg, S., (1994). "A Modular Approach to Fault-Tolerant Broadcasts and Related Problems".
- Kshemkalyani, A. D., Singhal, M., (2008). "Distributed computing: principles, algorithms, and systems". Cambridge University Press.
- Mattoso, M., Dias, J., Ocaña, K., et al., (2015). "Dynamic steering of HPC scientific workflows: A survey". *Future Generation Computer Systems*, 46, pp. 100–113.
- NACAD, (2015). *SGI Altix ICE 8400 (Uranus)*. Acesso em 08 de 2015, disponível em <http://www.nacad.ufrj.br/en/recursos/sgi8200>
- Oliveira, D., Costa, F., Silva, V., et al., (2014). "Debugging Scientific Workflows with Provenance: Achievements and Lessons Learned". *Proceedings of the XXIX Brazilian Symposium on Databases*. Curitiba, Paraná.
- Özsu, M. T., Valduriez, P., (2011). "Principles of Distributed Database Systems (3 ed.)". New York: Springer.
- Raicu, I., Foster, I., Zhao, Y., (2008). "Many-task computing for grids and supercomputers". *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, pp. 1-11.
- Souza, F., (2015). "Controlling the Parallel Execution of Workflows Relying on a Distributed Database".
- Souza, R., Silva, V., Neves, L., et al., (2015). "Monitoramento de Desempenho usando Dados de Proveniência e de Domínio durante a Execução de Aplicações

Científicas". *Anais do XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*. Recife, PE.