%% modeloABNT2.tex, v1.0 athila

%% Copyright 2013 by Athila e Monaro

%%

%% Este trabalho é uma adequação das normas de dissertações e teses

%% da Universidade de São Paulo - USP de acordo com a Norma ABNT

%%

%%

%% Quaisquer dúvidas favor enviar e-mail para:

%%

%% athila.uno@gmail.com ou

%% renato.monaro@gmail.comsh

%%

% ------------------------------------------------------------------------

% ------------------------------------------------------------------------

% eesc: Modelo de Trabalho Acadêmico (tese de doutorado, dissertação de

% mestrado e trabalhos monográficos em geral) em conformidade com

% ABNT NBR 14724:2011. Esta classe estende as funcionalidades da classe

% abnTeX2 elaborada de forma a adequar os parâmetros exigidos pelas

% normas USP e do departamento de elétrica da Escola de Engenharia

% de São Carlos - USP.

% ------------------------------------------------------------------------

% ------------------------------------------------------------------------

% ------------------------------------------------------------------------

% Opções:

% tesedr: Formata documento para tese de doutorado

% qualidr: Formata documento para qualificação de doutorado

% dissertmst: Formata documento para dissertação de mestrado

% qualimst: Formata documento para qualificação de mestrado

% ------------------------------------------------------------------------

\documentclass[qualimst]{eesc}

% ---

% PACOTES

% ---

% ---

% Pacotes fundamentais

% ---

\RequirePackage[applemac]{inputenc}

\usepackage{cmap} % Mapear caracteres especiais no PDF

\usepackage{lmodern} % Usa a fonte Latin Modern

\usepackage{makeidx} % Cria o indice

\usepackage{hyperref} % Controla a formação do índice

\usepackage{lastpage} % Usado pela Ficha catalográfica

\usepackage{indentfirst} % Indenta o primeiro parágrafo de cada seção.

\usepackage{nomencl} % Lista de simbolos

\usepackage{graphicx,url} % Inclusão de gráficos

% ---

% ---

% Pacotes adicionais, usados apenas no âmbito do Modelo eesc

% ---

\usepackage{lipsum} % para geração de dummy text

\usepackage[printonlyused]{acronym}

\usepackage[table]{xcolor}

% ---

\usepackage[brazilian,hyperpageref]{backref} % Paginas com as citaÁıes na bibl

\usepackage[alf]{abntex2cite} % CitaÁıes padr„o ABNT

% ---

% Informações de dados para CAPA e FOLHA DE ROSTO

% ---

%

% Título:

% 1. Título em português

% 2. Título em inglês

\titulo{\emph{Middleware} para Coordenar Tolerância a Falhas e Elasticidade}{Título em inglês}

%

% Autor:

% 1. Nome completo do autor

% 2. Formato de nome para bibliografia

\autor{Eduardo Henrique Ferreira Mendes Teixeira}{H. F. M. Teixeira, Eduardo}

%

% Cidade

\local{Brasília}

% Ano de defesa

\data{2013}

% Área de concentração da pesquisa

\areaconcentracao{Infraestrutura de TI}

% Nome do orientador

\orientador{Prof.\textsuperscript{a} Dr.\textsuperscript{a} Aletéia Patrícia Favacho de Araújo}

% Nome do coorientador

% ---

% ---

% ConfiguraÁıes de aparÍncia do PDF final

% alterando o aspecto da cor azul

\definecolor{blue}{RGB}{41,5,195}

% informaÁıes do PDF

\makeatletter

\hypersetup{

%pagebackref=true,

pdftitle={\@title},

pdfauthor={\@author},

pdfsubject={\imprimirpreambulo},

pdfcreator={LaTeX with abnTeX2},

pdfkeywords={abnt}{latex}{abntex}{abntex2}{relatÛrio tÈcnico},

colorlinks=true, % false: boxed links; true: colored links

%linkcolor=blue, % color of internal links

filecolor=magenta, % color of file links

urlcolor=blue,

bookmarksdepth=4

}

\makeatother

% ---

% ---

% compila o indice

% ---

\makeindex

% ---

% ---

% Compila a lista de abreviaturas e siglas

% ---

\makenomenclature

% ---

% ---

% Inserir ficha catalográfica

%

% Caso o comando \inserirfichacatalografica seja definido, a ficha catalográfica

% será inserida atrás da folha de rosto. Caso contrário a página será deixada em

% branco.

%

% CUIDADO: Esta opção deve ser preenchida antes do comando \maketitle

% ---

%\inserirfichacatalografica{fichaCatalografica.pdf}

% ---

% ---

% Inserir folha de aprovação

%

% Caso o comando \inserirfolhaaprovacao seja definido, a a folha de aprovação

% será inserida. Além disso, conforme Resolução CoPGr 5890, as informações

% de rodapé são inseridas apropriadamente na folha de rosto.

%

% CUIDADO: Esta opção deve ser preenchida antes do comando \maketitle

% ---

%\inserirfolhaaprovacao{folhaAprovacao.pdf}

% ---

% ----

% Início do documento

% ----

\begin{document}

% ----------------------------------------------------------

% ELEMENTOS PRÉ-TEXTUAIS

% ----------------------------------------------------------

\pretextual

% ---

% Insere Capa, Folha de rosto, Ficha catalográfica (se inserida)

% e folha de aprovação (se inserida).

% ---

\maketitle

% ---

% Dedicatória

% ---

%\imprimirdedicatoria{Este trabalho é dedicado à minha família e amigos.}

% ---

% ---

% Agradecimentos

% ---

%\imprimiragradecimentos{

%Os agradecimentos principais são direcionados à minha filha Heloísa e esposa Vaneide, ao meu irmão Benício, minha mãe Socorro, irmã Silvana, Vladmir e a todos os amigos.

%Agradecimentos especiais à equipe de desenvolvimento de software da Sysofit pelo apoio e por fazer parte deste projeto muito ambicioso.

%}

% ---

% ---

% Epígrafe

% ---

%\imprimirepigrafe{

% ``Somos essencialmente profissionais do sentido. Educamos, \\

% quando ensinamos com sentido. Educar é impregnar de sentido\\

% a vida. A profissão docente está centrada na vida, no bem querer.''\\

% (Prof. Gilberto Teixeira)

%}

% ---

% ---

% RESUMO e ABSTRACT

% ---

% Resumo em português

\begin{resumo}{\emph{Middleware}, Tolerância a Falhas, Elasticidade, \emph{Cluster}}

Este trabalho propõe um \emph{middleware} para implementar tolerância a falhas e elasticidade em um \emph{cluster} de alta performance e alta disponibilidade. Assim sendo, espera-se que o \emph{middleware} proposto seja capaz de diminuir o número de servidores necessários para processar as filas de mensagens de um sistema distribuído e, consequentemente, economizar recursos computacionais para manutenções evolutivas e corretivas. Para isso é necessária uma arquitetura que seja elástica o suficiente para se adaptar ao crescimento da fila de requisições de forma que as mensagens não se acumulem, e que seja tolerante a falhas para que eventuais paradas do sistema, por queda ou falha dos serviços, não impacte na operacionalidade do \emph{cluster}.

\end{resumo}

% ---

% ---

% inserir lista de ilustrações

% ---

%\listailustracoes

% ---

% ---

% inserir lista de tabelas

% ---

%\listatabelas

% ---

% ---

% inserir lista de abreviaturas e siglas

% ---

%\listasiglas{abrev/Abreviaturas}

% ---

% ---

% inserir o sumario

% ---

\sumario

% ---

% ----------------------------------------------------------

% ELEMENTOS TEXTUAIS

% ----------------------------------------------------------

\mainmatter

% ----------------------------------------------------------

% Introdução

% ----------------------------------------------------------

\chapter[Introdução]{Introdução}

A computação distribuída consiste em integrar o poder computacional de diversos computadores por meio de uma rede de computadores. A computação distribuída lida com os aspectos da computação relacionados ao acesso e troca de informações através de múltiplos elementos de processamento e plataformas conectadas por redes de comunicação \cite{AJAY}.

Um sistema de computação distribuída consiste de processos distintos que não compartilham memória e se comunicam assincronamente por passagem de mensagem ou canais de comunicação \cite{AJAY}. Ela começou a surgir com os avanços e as evoluções nas áreas de redes e hardware, e com a diminuição do custo de equipamentos. Assim, estes fatores fizeram com que a computação distribuída com alta performance e tolerância a falhas se tornasse uma realidade.

Entre os vários tipos de arquitetura para realizar a computação distribuída existe o \emph{cluster}, que caracteriza-se por possuir uma homogeinedade de hardware e software entre todos os nós que estão conectados \cite{TANENBAUM\_STEEN}.

O \emph{cluster} consiste em um conjunto de computadores conectados em uma rede local para trabalharem em conjunto, que são vistos como um único sistema, apesar de cada nó rodar sua própria instância de sistema operacional. O \emph{cluster} é usado para melhorar a performance, a disponibilidade e o custo de aplicações por meio do uso de vários computadores.

A computação em \emph{cluster} utiliza um gerenciamento centralizado para prover serviços, e trata os nós disponíveis como servidores compartilhados, tornando esta abordagem transparente para o usuário da aplicação.

%\subsection{\emph{Grid}}

%

%É um modelo computacional que pode alcançar uma alta taxa de processamento através da divisão de tarefas entre vários servidores, em uma rede local ou de longa distância, que formam uma máquina virtual. Esses processos podem ser executados em momentos em que estes servidores estão ociosos, evitando o desperdício de processamento da máquina utilizada.

%

%Uma possível arquitetura para o gride possue as seguintes camadas:

%

%\begin{itemize}

% \item \textbf{\emph{Fabric Layer}}\\

% Camada mais baixa que provê funções para consultar o estado e as capacidades de um recurso.

%

% \item \textbf{Camada de Conectividade}\\

% Consiste em protocolos de comunicação para suportar as transações no \emph{grid} que usam múltiplos recursos. Contém os protocolos de segurança responsáveis por autenticar usuários e recursos.

%

% \item \textbf{Camada de Recursos}\\

% É responsável por gerenciar um único recurso. Usa funções providas pela camada de conectividade e chama diretamente as interfaces disponibilizadas pela \emph{fabric layer}.

%

% \item \textbf{Camada Coletiva}\\

% Lida com a manipulação de acesso a múltiplos recursos e tipicamente consiste de serviços para descobrimento de recursos, alocação e agendamento de tarefas para múltiplos recursos e replicação de dados. Consiste de diferentes protocolos para diferentes propósitos.

%

% \item \textbf{Camada de Aplicação}\\

% Consiste das aplicações que irão operar dentro de uma organização virtual e que usarão o ambiente de computação em \emph{grid}.

%

%\end{itemize}

%

%As camadas de conectividade, coletiva e de recursos juntas forma a camada de \emph{middleware}\footnote{Camada de software constituída por módulos com APIs (\emph{Application Program Interfaces}) de alto nível que proporcionam a sua integração com aplicações desenvolvidas em diversas linguagens de programação e interfaces de baixo nível e que permitem a abstração de programação e mascaramento da hetereogeinedade.} do \emph{grid}. Estas camadas juntas provêem acesso para o gerenciamento de recursos que são potencialmente espalhados através de múltiplos sítios.

%

%\subsection{\emph{Cloud}}

%O termo \emph{cloud computing} (comutação em nuvem) veio da área de telecomunicações quando os provedores começaram a usar os serviços de VPNs (\emph{Virtual Private Networks}) para comunicação de dados \cite{JADEIA\_MODI}. A computação em nuvem lida com computação distribuída, software, acesso a dados e serviços de armazenamento que não requerem conhecimento por parte dos usuários finais sobre a localização física e dos sistemas que estão provendo os serviços. Existem quatro tipos de implementações de nuvens: pública; privada; híbrida; comunitária.

%

%Na nuvem pública os usuários acessam os serviços via \emph{web browsers} e pagam somente pelo tempo que usarem (\emph{pay-per-use}), reduzindo os custos com despesas de TI (Tecnologia da Informação). Porém, estas nuvens são inseguras se comparadas com outros modelos de nuvens. Uma nuvem privada opera dentro da organização e sua principal vantagem é que se torna fácil gerenciar a segurança, manutenções, desenvolvimento e uso. Uma nuvem privada é conectada a um ou mais serviços externos à organização. É uma maneira mais segura de controlar os dados e as aplicações e permitir que outras partes acessem as informações pela \emph{internet}. A nuvem híbrida é uma combinação das nuvens pública e privada. A nuvem comunitária é usada quando muitas organizações constroem e compartilham a infraestrutura, requerimentos e políticas para prover serviços por uma das organizações da comunidade.

%

%Entre as vantagens do uso de nuvens estão o fácil gerenciamento, custo reduzido, serviços ininterruptos, gerenciamento de desastres e economia de energia. Entre as desvantagens estão a privacidade, segurança, tempo de replicação e confiabilidade.

%

%A computação em nuvem provê serviços através de seis maneiras:

%

%\begin{itemize}

% \item \textbf{IaaS - \emph{Infrastructure as a Service}}\\

% Quando se utiliza uma porcentagem de um servidor, geralmente com configuração que se adeque à sua necessidade.

%

% \item \textbf{PaaS - \emph{Plataform as a Service}}\\

% Utiliza apenas uma plataforma como um banco de dados ou \emph{web-service} como é o caso do \emph{Windows Azure}.

%

% \item \textbf{DevaaS - \emph{Development as a Service}}\\

% As ferramentas de desenvolvimento tomam forma na nuvem como ferramentas compartilhadas, ferramentas de desenvolvimento \emph{web-based} e serviços baseados em \emph{mashup}\footnote{É um site personalizado ou uma aplicação \emph{web} que usa conteúdo de mais de uma fonte para criar um novo serviço completo.}.

%

% \item \textbf{SaaS - \emph{Software as a Service}}\\

% Uso de software para utilização em formato \emph{web} como no \emph{Google Docs}, \emph{Microsoft SharePoint Online}.

%

% \item \textbf{CaaS - \emph{Communication as a Service}}\\

% Uso de uma solução de comunicação unificada hospedada em um \emph{Data Center} do provedor ou fabricante como o \emph{Microsoft Lync}.

%

% \item \textbf{DBaaS - \emph{Data Base as a Service}}\\

% Utiliza servidores de banco de dados como serviço.

%

%\end{itemize}

%

\section{Definição do Problema}

O sistema distribuído da Sysofit, empresa na qual ocorrerá a implementação e análise do \emph{middleware} proposto, funciona em um \emph{cluster} de alta performance composto por 58 servidores. As manutenções evolutivas do sistema (novas versões) duram cerca de 3 horas em função da quantidade de servidores que devem ser atualizados. Estas atualizações devem ser realizadas com aviso prévio aos seus clientes e agendamento de \emph{downtime}\footnote{Períodos de tempo em que um sistema está indisponível por conta de rotinas de manutenção.}.

Além disso, o sistema também fica inoperante durante problemas de funcionamento por queda ou falha no tratamento de mensagens das filas, até que o problema seja encontrado e resolvido. Ambas as situações levam a uma diminuição da qualidade dos serviços prestados aos clientes da Sysofit.

Assim, a conexão com os servidores para o envio e o recebimento de mensagens, e posições consomem quase 100\% dos 4 núcleos da CPU de cada nó do \emph{cluster} durante um ciclo de conexão, como pode ser observado na Figura \ref{fig:machine\_cpu\_4cores}.

\begin{figure}[h!]

\centering

\includegraphics[width=.80\textwidth,height=.30\textheight]{machine\_cpu\_4cores.png}

\caption{Uso de CPU de um Servidor Durante 1 Ciclo de Conexão de 80 Segundos.}

\label{fig:machine\_cpu\_4cores}

\end{figure}

Dessa forma, após o término desse ciclo de conexão, que é realizado rapidamente (menos de 10 segundos), o servidor fica ocioso esperando um novo ciclo completar o período de 80 segundos. Essa situação pode ser observada na Figura \ref{fig:machine\_cpu\_1h}, onde os servidores utilizam a CPU em rajadas muito rápidas e, em seguida, ficam ociosos a maior parte do tempo. Esta situação representa um uso ineficiente de recursos, pois os servidores ficam 87,5\% do tempo com ociosidade de consumo de CPU e apenas 12,5\% com pico de consumo de CPU.

\begin{figure}[h!]

\centering

\includegraphics[width=.80\textwidth,height=.30\textheight]{machine\_cpu\_1h.png}

\caption{Uso de CPU de um Servidor Durante 1 hora}

\label{fig:machine\_cpu\_1h}

\end{figure}

Nesse cenário, nota-se que é necessário implementar um \emph{middleware} eficiente para o \emph{cluster} da Sysofit, a fim de garantir o uso eficiente dos recursos por meio da técnica de elasticidade, e uso seguro e confiável por meio das técnicas de tolerância a falhas. Assim sendo, este trabalho propõe implementar um \emph{middleware} que garanta elasticidade e tolerância a falhas.

\section{Objetivos}

\subsection{Objetivo Geral}

O objetivo geral deste trabalho é desenvolver um \emph{middleware} que suporte elasticidade de forma transparente à aplicação distribuída e que seja tolerante a falhas.

\subsection{Objetivos Específicos}

Para cumprir o objetivo geral, este trabalho tem os seguintes objetivos específicos:

\begin{itemize}

\item \textbf{Levantamento do estado da arte através da análise crítica dos artigos científicos recentes (a partir de 2010) referentes aos temas tolerância a falhas e elasticidade.}

\item \textbf{Estudo das abordagens mais relevantes para serem utilizadas na implementação das características de tolerância a falhas e elasticidade no \emph{middleware} que será proposto.}

\item \textbf{Testes e análises do \emph{midleware} desenvolvido para posterior migração e adequação no ambiente real da Sysofit.}

\item \textbf{Publicação de artigos científicos contendo descobertas relevantes e os resultados dos experimentos realizados.}

\end{itemize}

%Isso não é objetivo específico. É detalhe de implementação!

%O objetivo deste trabalho é desenvolver um \emph{middleware} gerenciado por um coordenador que possui, por processo, as tabelas de consumo de CPU, memória, número de mensagens nas filas e vazão de consumo de itens pelos \emph{middlewares} escravos. Desta forma, o \emph{middleware} suportará elasticidade de forma transparente à aplicação distribuída que esteja rodando no \emph{cluster} através do envio, pelo coordenador, de solicitações de inicialização e finalização de processos para os servidores de acordo com a vazão de consumo requerida para que não ocorram enfileiramentos e nem desperdício de recursos com processos desnecessários. Ou seja, o \emph{middleware} coordenador realizará o cálculo de número de processos necessários para que o processamento atenda a demanda requisitada.

%

% Assim não será mais necessário fazer qualquer configuração estática e manual sob o risco de ser mal dimensionada, e consumir recursos desnecessários. Para suportar tolerância a falhas com alta disponibilidade, em caso de falha ou queda de processos do sistema, o \emph{middleware} coordenador automaticamente inicia um novo processo associado às filas do antigo processo e registra o ocorrido em seus \emph{logs}.

%

%Caso o sistema chegue em seu limite de processamento pela detecção de saturação de CPU e crescimento das filas o \emph{middleware} coordenador gerará alarmes em seus \emph{logs}. Neste caso o sistema encontrou o seu limite de funcionamento de acordo com o dimensionamento de servidores disponibilizado e tende a enfileirar mensagens durante um tempo maior do que o definido nos parâmetros de qualidade.

%

%Para se alcançar a transparência de elasticidade os processos da aplicação distribuída devem ter baixo acoplamento, ou seja, não podem possuir qualquer tipo de dependência de configuração manual para seu funcionamento. Isso é necessário pois o parâmetro que o \emph{middleware} coordenador usará para reconhecer que um processo não está conseguindo tratar as demandas de requisições é o aumento das filas de mensagens associadas a ele.

%

%Assim a aplicação distribuída da Sysofit poderá ser dimensionada corretamente em número de servidores necessários para tratar as demandas de processamento de mensagens sem comprometer a qualidade de serviço imposta por seus clientes. Consequentemente, isso reduzirá os custos com o hardware necessário para rodar as aplicações, pois o dimensionamento real, sob demanda, será calculado dinamicamente pelo próprio sistema.

\section{Proposta}

Neste trabalho é proposta a implementação de um \emph{middleware} eficiente e transparente que garanta a elasticidade e a tolerância a falhas, objetivando obter um melhor desempenho do sistema distribuído. Dessa forma, o dimensionamento de processos utilizados será feito de forma a atender a demanda real de processamento, necessária para que não haja enfileiramento de mensagens e, consequentemente, atrasos na operacionalidade do sistema. Assim sendo, será necessário desenvolver um \emph{middleware} para suportar elasticidade a fim de conter o crescimento das filas, e suportar também tolerância a falhas para ser resistente a quedas e falhas.

%O \emph{middleware} a ser desenvolvido deve ser elástico de forma a conter o crescimento das filas por meio do aumento da vazão, alcançado com o paralelismo de consumo. Da mesma forma, quando as filas estiverem com um número de mensagens menor, o \emph{middleware} deve finalizar os processos desnecessários. Em caso de queda ou falha de processos o \emph{middleware} deverá reinicializar automaticamente os processos que tiveram problemas no tratamento da mensagem. Caso o problema persista, a mensagem deve ser redirecionada para um repositório persistente para análise posterior para não comprometer o processamento das demais mensagens da fila.

\section{Estrutura do Trabalho}

O restante deste trabalho está dividido em cinco capítulos. O Capítulo 2 aborda os sistemas distribuídos, os \emph {middlewares}, os tipos de ambientes de computação distribuída e, em seguida, o cenário da Sysofit. O próximo capítulo, Capítulo 3, critica e relaciona os principais artigos científicos em relação aos temas tolerância a falhas e elasticidade. Em seguida, no Capítulo 4, é proposta uma solução com base na literatura pesquisada. Para finalizar, o Capítulo 5 apresenta um cronograma de trabalho para a pesquisa e, o Capítulo 6 encerra com as conclusões pertinentes.

\chapter{Fundamentação Teórica}\label{cap\_exemplos}

\chapterprecishere{Este capítulo traz uma breve apresentação dos conceitos relacionados ao desenvolvimento deste trabalho. Também é apresentado o cenário da Sysofit, ao qual este projeto de pesquisa se destina.}\index{sinopse de capitulo}

\section{Sistemas Distribuídos}

É uma coleção de computadores autônomos e independentes que se apresenta ao usuário como um sistema único e consistente \cite{TANENBAUM\_STEEN}. Segundo \cite{COULOURIS}, é uma coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema como hardware, software e dados.

O processamento dos sistemas distribuídos utiliza computação paralela realizada por duas ou mais entidades independentes, conectadas em rede e que cooperam entre si dividindo tarefas para resolver um problema. Para isso, os sistemas distribuídos utilizam uma camada de software chamada \emph{middleware} que torna o seu desenvolvimento mais fácil e ágil. Esta camada proporciona aumento da abstração para os desenvolvedores de aplicações com relação à comunicação, execução de requisições, escalabilidade e hetereogeinedade.

\subsection{\emph{Middleware}}

O \emph{Middleware} é representado por processos ou objetos em um conjunto de computadores que interagem entre si para implementar a comunicação e o compartilhamento de recursos para aplicações distribuídas \cite{COULOURIS}. Ele eleva o nível das atividades de comunicação dos programas aplicativos por meio do suporte a abstrações como invocação remota de métodos, comunicação entre grupos de processos, notificação de eventos, replicação de dados compartilhados e transmissão de dados multimídia em tempo real.

Nesse contexto, o \emph{middleware} é uma camada de software que provê abstração de programação, bem como, o mascaramento da hetereogeinedade das redes, hardware, sistemas operacionais e linguagens de programação \cite{COULOURIS}. Atualmente, para se desenvolver um sistema distribuído em \emph{cluster} utiliza-se, geralmente, um \emph{middleware} como CORBA ({\emph{Common Object Request Broker Architeture} \cite{TANENBAUM\_STEEN}), RMI ({\emph{Remote Method Invocation} \cite{COULOURIS}), RPC (\emph{Remote Procedure Call} \cite{COULOURIS}), DCOM (\emph{Distributed Component Object Model} \cite{AJAY}), ou desenvolve-se uma API\footnote{\emph{Application Program Interface} - uma interface consistindo de chamadas de biblioteca \cite{TANENBAUM\_STEEN}.} proprietária tornando a aplicação dependente desta camada.

Assim sendo, muitas aplicações distribuídas dependem inteiramente dos serviços providos pelo \emph{middleware} disponível para suportar suas necessidades de comunicação e compartilhamento de dados. Muito tem sido feito para simplificar a programação de sistemas distribuídos através do desenvolvimento do suporte do \emph{middleware}, mas alguns aspectos da dependabilidade dos sistemas requerem suporte no nível de aplicação.

Para o correto funcionamento de programas distribuídos existe a dependência de checagem, os mecanismos de correção de erros, as medidas de segurança em muitos níveis, alguns dos quais requerem acesso aos dados do espaço de endereçamento da aplicação. Qualquer tentativa de realizar essas checagens dentro do \emph{middleware}, somente vai garantir uma parte do funcionamento correto requerido, podendo gerar duplicação de trabalho nas aplicações e desperdício do esforço de programação por meio da adição de complexidade e redundância desnecessárias, que prejudicam a performance do sistema distribuído.

Os \emph{middlewares} são projetados para funcionar de acordo com as peculiaridades específicas de cada plataforma distribuída, fazendo com que sejam customizados para as características de cada uma destas arquiteturas.

Dessa forma, o \emph{middleware} facilita a implementação de aplicações distribuídas em qualquer plataforma. Nesse contexto, a próxima seção descreve a infraestrutura distribuída chamada de \emph{cluster}, pois foi a primeira plataforma distribuída, e fortemente ainda usada.

%A seguir veremos o funcionamento da arquitetura de sistemas distribuídos denominada \emph{cluster}.

\subsection{\emph{Cluster Computing}}

Uma importante classe de sistemas distribuídos são os usados para as tarefas de computação de alta performance \cite{TANENBAUM\_STEEN}. Na computação em \emph{cluster} o hardware consiste em uma coleção de estações de trabalho similares conectadas em redes locais de alta velocidade, em que cada nó executa o mesmo sistema operacional. A situação se torna um pouco diferente no caso de computação em \emph{grid} \cite{TANENBAUM\_STEEN}, pois essa plataforma consiste de sistemas distribuídos que são, geralmente, construídos de forma que cada sistema pode executar em domínios administrativos diferentes.

A computação em \emph{cluster} é bastante usada para programação paralela, na qual um único programa (de computação intensiva) roda em paralelo em múltiplas máquinas \cite{TANENBAUM\_STEEN}. Um exemplo de plataforma de \emph{cluster} é o \emph{Beowulf} \cite{TANENBAUM\_STEEN}.

O \emph{cluster} \emph{Beowulf} consiste de uma coleção de nós que são controlados e acessados por meio de um único nó mestre. O mestre tipicamente manipula a alocação dos nós para um programa paralelo, mantém uma fila em lote de tarefas submetidas e provê uma interface para os usuários do sistema. Assim, o mestre roda um \emph{middleware} necessário para a execução dos programas e gerenciamento do \emph{cluster}, enquanto os nós computacionais necessitam apenas de um sistema operacional.

Existem vários tipos de \emph{cluster}, entre eles destacam-se:

\begin{itemize}

\item \textbf{\emph{Cluster} de Alto Desempenho}\\

Conhecido também como \emph{cluster} de alta performance, funciona permitindo que ocorra uma grande carga de processamento com um alto volume de \emph{gigaflops} em computadores comuns. %Um exemplo desse tipo de \emph{cluster} é o \emph{Beowulf}\footnote{Um \emph{cluster} de computação paralela de alta performance cujo hardware é composto por computadores baratos como os PCs (\emph{Personal Computers}).}.

\item \textbf{\emph{Cluster} de Alta Disponibilidade}\\

São \emph{clusters} cujos sistemas conseguem permanecer ativos por um longo período de tempo em condição de uso, sem a necessidade de reinício, ou seja, de maneira ininterrupta. Além disso, conseguem detectar eficientemente erros, proporcionando que os mesmos sejam tolerantes a falhas.

\item \textbf{\emph{Cluster} para Balanceamento de Carga}\\

Esse tipo de \emph{cluster} tem como função controlar a distribuição equilibrada do processamento por meio de um monitoramento constante na sua comunicação e em seus mecanismos de redundância, evitando que ocorra uma interrupção no seu funcionamento em caso de falha.

\end{itemize}

Uma parte importante do \emph{middleware} é formada por bibliotecas para a execução de programas em paralelo. Muitas dessas bibliotecas proveêm efetivamente facilidades de comunicação avançada baseada em mensagem, mas não são capazes de suportar manipulação de falhas de processos ou segurança \cite{TANENBAUM\_STEEN}.

Como uma alternativa a essa organização hierárquica, uma abordagem simétrica é seguida pelos sistemas MOSIX\footnote{Sistema operacional para \emph{multi-clusters} que incorpora distribuição dinâmica da carga de trabalho \cite{TANENBAUM\_STEEN}.}, que tenta prover a imagem de um \emph{cluster} de sistema único. Assim, o \emph{cluster} oferece ao processo transparência de distribuição parecendo ser um único computador. Prover essa imagem em todas as circunstâncias é impossível. No caso do MOSIX um alto grau de transparência é provido permitindo aos processos dinamicamente e preemptivamente migrarem entre nós que fazem parte do \emph{cluster}. A migração de processos permite ao usuário iniciar uma aplicação em qualquer nó, e depois do qual ele pode transparentemente ser transferido para outros nós, fazendo uso eficiente dos recursos \cite{TANENBAUM\_STEEN}.

%\subsection{\emph{Grid}}

%

%São sistemas de computadores conectados em rede onde as máquinas utilizam os mesmos recursos coletivamente. Geralmente consiste de um computador principal que distribui informação e tarefas para um grupo de computadores responsável por realizar uma tarefa em comum.

%

%Uma característica do \emph{cluster} é a homogeinedade, onde os computadores são os mesmos, tem os mesmos sistemas operacionais e estão conectados na mesma rede. Em contraste a isso os computadores em \emph{grid} tem um alto grau de hetereogeinedade, onde nenhuma suposição é feita sobre o hardware, sistema operacional, redes, domínios administrativos ou políticas de segurança.

%

%A questão chave na computação em \emph{grid} é que os recursos de diferentes organizações trabalham em conjunto para permitir a colaboração de um grupo de pessoas ou instituições e os serviços são realizados na forma de uma organização virtual. Assim as pessoas pertencentes a uma mesma organização virtual tem direitos de acesso aos recursos que são providos para aquela organização. Estes recursos consistem de computadores servidores, facilidades de armazenamento e bancos de dados.

%

%A maior parte do software para realizar computação em \emph{grid} são para prover acesso de recursos a partir de diferentes domínios administrativos e não somente para os usuários e aplicações que pertencem a uma organização virtual específica. Por esta razão o foco de implantação é frequentemente em questões arquiteturais.

%

%

%

%\subsection{\emph{Cloud}}

%

%O uso de computação em nuvem é uma tendência recente na área de TI que move toda a infraestrutura de computação para grandes centros de processamento de dados capazes de processar grandes quantidades de informações sob demanda e de vários clientes.

%

%O principal objetivo da computação em nuvem é fazer o melhor uso dos recursos distribuídos, combinando-os para alcançar uma alta vazão, estando apto para resolver problemas de computação de larga escala. Para isso a computação em nuvem tem que lidar com problemas como: virtualização, escalabilidade, interoperabilidade, qualidade de serviço através das nuvens chamadas de pública, privada, híbrida e comunitária.

%

%Na computação em nuvem os usuários podem acessar dados, aplicações e qualquer serviço independente da localização, sendo a infraestrutura fornecida por uma terceira parte com a ajuda da \emph{internet}. Possue um custo reduzido pelo fato da computação necessária possuir picos de processamento ocasionais e precisa de menos habilidades em TI para ser implementada. Serviços confiáveis que favorecem a continuidade dos negócios e recuperação de desastres são obtidos através de redundância, porém paradas e interrupções podem ocorrer sem que nada possa ser feito a respeito.

%

%O compartilhamento de recursos garantem um uso eficiente da infraestrutura e a manutenção se torna mais fácil, pois não é mais necessário instalar as aplicações nos computadores de cada usuário. As facilidades de pagar por uso torna mais fácil o faturamento. A performance é facilmente monitorável e favorável à escalabilidade. Questões de segurança são resolvidas de uma maneira melhor, pois os provedores dedicam recursos que os clientes não poderiam pagar, porém a confidencialidade é um importante tópico que faz com que ainda haja atraso na adoção dessa tecnologia.

%

\section{Cenário Sysofit }

O sistema distribuído da Sysofit monitora o posicionamento (latitude e longitude) de mais de 365.000 veículos. Ele utiliza um \emph{cluster} com 58 servidores (232 núcleos e 464 \emph{gigabytes} de memória) e necessita de configuração e inicialização manuais. Além disso, ele possui dez bases de dados \emph{Microsoft SQLServer}, totalizando 1.7 \emph{terabytes} de informação a partir de 15.031 contas de clientes. Cada base possui um conjunto de servidores associado estaticamente, e armazena um histórico de 6 meses de dados de posições (2.098 bilhões), conforme mostrado na Tabela \ref{tab:tabelabd}.

Neste ambiente, cada servidor possue 19 serviços e realiza a obtenção de dados por meio de \emph{polling}\footnote{Requisição de amostras de estado de maneira síncrona por um cliente.} a cada 80 segundos em dois servidores: um responsável pela comunicação via satélite e outro via rede celular.

\begin{table}[!htpb]

\centering

\caption{Bases de Dados com Histórico de 6 Meses do Sistema Distribuído da Sysofit.}

\begin{center}

\begin{tabular}{|l|c|c|c|c|c|}

\hline

Base de Dados &Tamanho (\emph{megabytes}) &Servidores &Posições &Contas &Veículos\\

\hline\hline

Instância 1 &294.941,50 &8 &460.005.267 &1.823 &57.730\\

\hline

Instância 2 &276.255,88 &11 &383.133.207 &4.141 &66.103\\

\hline

Instância 3 &239.518,00 &4 &236.692.312 &1.904 &63.406\\

\hline

Instância 4 &225.487,19 &10 &260.621.400 &3.095 &70.966\\

\hline

Instância 5 &137.003,75 &3 &122.688.469 &1.056 &32.145\\

\hline

Instância 6 &36.864,00 &2 &36.609.157 &221 &6.491\\

\hline

Instância 7 &9.695,75 &1 &178.696 &42 &249\\

\hline

Instância 8 &161.418,38 &5 &222.614.618 &1.309 &22.354\\

\hline

Instância 9 &212.302,75 &8 &309.203.587 &1.791 &33.094\\

\hline

Instância 10 &126.976,00 &6 &66.800.758 &1.553 &13.069\\

\hline\hline

\textbf{Total} &\textbf{1.720.463,20} &\textbf{58} &\textbf{2.098.547.471} &\textbf{15.031} &\textbf{365.358}\\

\hline

\end{tabular}

\label{tab:tabelabd}

\end{center}

\end{table}

Cada servidor executa um sistema distribuído baseado em um coordenador que realiza IPC (\emph{Interprocess Communication}) com os demais serviços por meio de \emph{sockets} TCP (\emph{Transmission Control Protocol}) e \emph{Message Queue}. Cada um desses serviços é especializado em uma funcionalidade como, por exemplo, o pedido de posicionamento (latitude e longitude), o controle de rotas e a troca de mensagens e comandos, entre outras.

No sistema distribuído da Sysofit, foi desenvolvida uma camada de objetos distribuídos replicados que faz \emph{cache} dos dados mais acessados do banco de dados na carga inicial do sistema, evitando que as consultas SQL, durante o funcionamento, onerem o servidor de banco, o que causaria lentidão de acesso pelos servidores de aplicação e pela interface de usuário. O sistema lida com esse problema enviando atualizações sempre para todas as cópias replicadas na medida em que elas são

realizadas no coordenador.

%\begin{figure}[!htb]

% \centering

% \mbox{%

% \subfigure[Uso de CPU do \emph{Cluster} Durante 24 horas.]{

% \includegraphics[width=.50\textwidth,height=0.21\textheight]{cluster\_cpu\_24hs.png}}\qquad

%\subfigure[Uso de CPU de um Servidor Durante 1 hora.]{

%\includegraphics[width=.50\textwidth,height=0.21\textheight]{machine\_cpu\_1h.png}}

%}

% \mbox{%

% \subfigure[Uso de CPU de um Servidor Durante 1 ciclo de conexão de 80 segundos.]{

% \includegraphics[width=.50\textwidth,height=0.21\textheight]{machine\_cpu\_4cores.png}}\qquad

%\subfigure[Uso de Memória de um Servidor Durante 1 hora.]{

% \includegraphics[width=.50\textwidth,height=0.21\textheight]{machine\_mem\_1h.png}}

% }

%\caption{Gráficos de Utilização de CPU e Memória do Sistema Distribuído em Análise.}

%\label{fig:cpumem}

%\end{figure}

Assim, toda vez que é feita uma manutenção corretiva ou evolutiva é necessário reiniciar todos os servidores do \emph{cluster} manualmente, além de realizar o balanceamento de carga estaticamente em todas as bases de dados. Além disso, quando o sistema sofre uma interrupção por queda de um conjunto de servidores, toda a operação de um subconjunto de clientes é afetada pelo tempo necessário que a carga inicial demorar. Para a manutenção evolutiva ou corretiva é necessário o agendamento de um \emph{downtime} que é reportado a todos seus clientes. Tanto a interrupção por queda como os \emph{downtimes} agendados afetam o SLA (\emph{Service Level Agreement}) da empresa com os seus clientes, prejudicando a operação funcional por conta da paralisação total ou parcial do sistema distribuído.

No ambiente do sistema distribuídos da Sysofit existe uma homogeinedade tanto na configuração de hardware dos servidores, da rede e dos sistemas operacionais. O presente trabalho foca então no uso de \emph{middlewares} para \emph{clusters} de alta performance e de alta disponibilidade, uma vez que essa é arquitetura presente.

O objetivo deste trabalho é diminuição do número de servidores para melhorar os tempos de manutenção corretiva e evolutiva, bem como, otimizar o uso de recursos de TI. A diminuição do número de servidores poderá gerar picos de crescimento nas filas de mensagens. Em função disso será implementado um \emph{middleware} que deverá suportar elasticidade para conter o crescimento das filas de mensagens cuja tendência mostre que o número de processos não possui vazão suficiente para que elas diminuam em tempo, que garanta o SLA com os clientes da Sysofit. O \emph{middleware} dará suporte à integridade sequencial, garantida por \emph{locks} no coordenador, ao acessar as regiões críticas de \emph{cache} replicado e compartilhado. Além disso, caso ocorram paradas de processos por queda ou falha, o \emph{middleware} será responsável pela resposta à necessidade de reiniciá-los caso o sistema ainda demande o seu funcionamento para que fique em um estado sem enfileiramentos.

Na Figura \ref{fig:vmscluster} é mostrada uma imagem das máquinas virtuais do \emph{cluster}, e o sumário de alocação de CPU e memória disponibilizados para os servidores.

\begin{figure}[h!]

\centering

\includegraphics[width=.99\textwidth,height=.45\textheight]{vms\_cluster.png}

\caption{Imagem do \emph{cluster} de 58 Servidores da Sysofit.}

\label{fig:vmscluster}

\end{figure}

% ---

% Capitulo Genérico

% ---

\chapter{Trabalhos Relacionados}

\chapterprecishere{Este capítulo analisa, critica e relaciona diversos artigos científicos associados à tolerância a falhas e à elasticidade.}

% ---

O estudo analítico dos trabalhos relacionados aos temas tolerância a falhas e elasticidade constitui uma importante ferramenta de apoio necessária para a realização de uma proposta de solução que se baseie nas abordagens que serão criticadas.

\section{Tolerância a Falhas}

Em \cite{WANG}\cite{CASTRO} são propostos mecanismos de gravação e interceptação de \emph{logs} para posterior recuperação. Em \cite{CASTRO}, além disso, também é utilizado um protocolo de recuperação de falhas transparente. No \emph{framework} HOG é utilizada uma arquitetura em que quando um nó falha não será afetada a integridade dos demais, e ao ocorrer queda ou travamento de algum processo a operacionalidade pode ser facilmente recuperada e realizada por outro grupo de serviços que acessam os mesmos dados \cite{HE}. Por outro lado, os trabalhos \cite{WANG}\cite{BICER}\cite{CASTRO}\cite{HE} são arquiteturas com mecanismos rápidos de recuperação de falhas.

Em \cite{MARTINS}\cite{CASTRO} é sugerido que as implementações de suporte às características de tolerância a falhas sejam feitas na biblioteca de comunicação onde será feito o \emph{middleware}, e que seja de forma transparente à aplicação.

Por outro lado, a abordagem de implementação de tolerância a falhas citada em \cite{TAMBE} é interessante quando se constrói um \emph{middleware} de propósito genérico, que atenda a vários tipos de projetos com diferentes requisitos de políticas de qualidade de serviço. Em soluções de \emph{middleware} onde não há necessidade de variações nas políticas de níveis de qualidade, a implementação de suporte a estas características é desnecessária.

A replicação de nós para a tolerância a falhas \cite{ABBES}\cite{MARTINS} tem um custo de memória necessária para manter os coordenadores auxiliares rodando \cite{ABBES}, e de comunicação em que as atualizações devem ser enviadas para as todas as réplicas de forma a manter o estado de \emph{cache} consistente \cite{ABBES}\cite{BICER}. Um outra desvantagem é que se a queda ocorrer por causa de um pacote que gera uma exceção, a mesma também ocorrerá nas \emph{k} réplicas.

A abordagem de \emph{snapshots}\footnote{É o estado de um sistema em um ponto particular no tempo.} periódicos de \cite{BICER}, mostra-se interessante por ter uma característica de recuperação rápida em caso de falha. Porém, o trabalho em grupo citado em \cite{BICER} tem a desvantagem do custo da comunicação de envio dos estados para os outros nós. É interessante o uso de \emph{snapshots} persistentes para que em caso de detecção de falha algum outro serviço assuma seu processamento, recuperando o último \emph{checkpoint} válido, porém sem envio dos estados para evitar a sobrecarga da comunicação.

Em \cite{ZHAO} é usada uma abordagem com o uso de UDP (\emph{User Datagram Protocol}) com uma proposta de protocolo com garantia de entrega confiável e ordenação total. Por outro lado, com o uso do TCP é possível detectar quando um nó falha por meio da queda da conexão \cite{BICER}\cite{MARTINS} e \emph{pings} periódicos com \emph{timeouts} \cite{MARTINS}.

\section{Elasticidade}

A abordagem citada em \cite{MA}, apesar de trazer transparência de paralelismo e elasticidade, é de difícil implementação pois requer conhecimentos de funcionamento da JVM e de codificação de baixo nível, além de tornar a aplicação dependente de ser desenvolvida na linguagem \emph{Java}. Porém, pode ser adaptada através da carga dinâmica de códigos que possuem funcionalidades de um grupo de serviços específico.

Assim, qualquer serviço responsável pelo consumo de mensagens pode processar qualquer tipo de mensagem através da sua subscrição em um coordenador e da carga da \emph{Dll}\footnote{\emph{Dynamic Link Library} - Biblioteca de vínculo dinâmico, é a implementação feita pela \emph{Microsoft} para o conceito de bibliotecas compartilhadas.} que possue o código que realizará este processamento. Dessa forma, porções de código podem ser ativadas para qualquer processo consumidor e estes podem estar executando em servidores distintos.

Em \cite{FANG} é citado que os \emph{middlewares} baseados em \emph{publishers/subscribers}\footnote{Padrão de projeto que realiza propagação de informações em uma via na qual o \emph{publisher} pode notificar um ou mais \emph{subscribers} quando o estado de um assunto muda \cite{SCHMIDT}.} tem tornado-se a base da coordenação e da cooperação de sistemas distribuídos, facilitando a disseminação de informação e integração de aplicações. Nessa arquitetura o \emph{middleware} recebe os eventos em um ambiente que é dinâmico e sujeito a mudanças irregulares e inesperadas, habilitando a automação do processamento dos dados pela transmissão da informação dos produtores (\emph{publishers}) para os consumidores (\emph{subscribers}). Um ponto atrativo nessa abordagem é que produtores e consumidores são totalmente desacoplados no tempo, espaço e controle de fluxo.

A abordagem citada em \cite{TRAN} é interessante quando a integridade sequencial no processamento de mensagens não é um requisito funcional. Assim, o sistema escala de forma transparente e automática \cite{MA}\cite{TRAN} através das mensagens nas filas que podem ser dimensionadas e redistribuídas conforme a vazão requerida, sem se preocupar com a ordem em que as mensagens são processadas. Porém, é uma desvantagem quando há a necessidade de manter a consistência sequencial baseada na ordem de entrega para os consumidores, para que os eventos gerados pelos serviços consumidores preservem a restrição temporal em que foram gerados.

Um sistema com múltiplas filas de mensagens \cite{TRAN}\cite{LI} e vários consumidores pode gerar assimetrias de distribuição nas atribuições feitas pelos \emph{publishers} aos \emph{subscribers} \cite{FANG}\cite{TRAN}\cite{LI}. Sendo assim, é necessário identificar uma correspondência de melhor candidato ao consumo de novas filas de forma a diminuir a latência de consumo e evitar a saturação de consumidores sobrecarregados. Assim, é possível determinar os melhores candidatos ao consumo de novos itens, baseado na maior vazão média identificada \cite{TRAN}\cite{LI} e no menor consumo de CPU médio \cite{LI}.

O número médio de novos itens também se mostra uma abordagem pró-ativa interessante para que o sistema possa se preparar previamente, subindo o número de processos necessários para suportar uma demanda de pico prevista baseada em históricos. Esta característica de provisionamento para alcançar a qualidade de serviço necessária representa uma otimização com a diminuição da latência do início de processos que seria feita no modo reativo.

Em \cite{SUGIKI} é sugerido um protótipo que adota um projeto baseado em \emph{microkernel}\footnote{É uma arquitetura de núcleo (kernel) de um sistema operacional cujas funcionalidades são quase todas executadas fora do núcleo ao invés de \emph{drivers} de dispositivos.} de sistemas operacionais (\emph{OS like middleware}) com abstração de recursos alcançada por objetos e funções, oferecendo um ambiente de \emph{script} como interface de programação. A abstração de recursos é explorada para prover acesso uniforme para vários tipos de recursos. São utilizadas técnicas de agendadores de sistemas operacionais para gerenciar e satisfazer a demanda por recursos eficientemente.

Nesse trabalho os recursos de máquinas virtuais são encapsulados como objetos de linguagem por meio de um conjunto de métodos de acesso uniforme para diferentes tipos de recursos. Assim, os recursos podem ser manipulados em um ambiente de \emph{script} de forma que os programas sejam desenvolvidos usando um paradigma orientado a objetos. Nesse caso são disponibilizadas funções que provêem paralelismo e distribuição para aumentar a escalabilidade usando eficientemente os núcleos disponíveis em cada máquina. A elasticidade é alcançada usando um agendador elástico, que trabalha no lado do usuário gerando clones de máquinas virtuais quando mais capacidade é requerida. Se o uso médio de CPU exceder um limite, clones de máquinas virtuais são iniciadas para prover elasticidade.

Segundo \cite{LEITNER} a computação elástica traz enormes vantagens para provedores de aplicações, incluindo economia de custos, prevenção de super-provisionamento de recursos de TI através do monitoramento de demanda e aquisição dos recursos estritamente requeridos pelas aplicações para alcançar um alvo de nível de qualidade. São indicadas métricas de alto nível que podem ser usadas para aumentar a elasticidade de aplicações em \emph{middlewares} existentes.

Os desenvolvedores de aplicação escrevem suas políticas de escalabilidade baseadas no significado de métricas de alto nível, como a capacidade de resposta de suas aplicações para certas operações de tempo crítico. Porém, métricas como número médio de novos dados processados por segundo e tempo médio (em milisegundos) necessário para processar um único novo item de dados tem uma relevância maior no domínio de negócio da aplicação e podem dar uma visão mais precisa da performance das aplicações do que a carga média de CPU, e também podem definir políticas de comportamento para a escalabilidade.

\cite{MA}\cite{TRAN}\cite{IMAI} exploram a transparência de elasticidade para a aplicação. \cite{SUGIKI}\cite{IMAI} usam o aumento ou diminuição do número de VMs de acordo com a demanda de carga exigida, que é medida pelo consumo de CPU \cite{SUGIKI}\cite{IMAI}\cite{LI}. Esta é uma abordagem interessante quando o objetivo é a contenção do crescimento de filas de mensagens, porém em sistemas distribuídos com características I/O \emph{bound}\footnote{Sistemas que fazem uso intensivo de Entrada/Saída (I/O).} pode se mostrar ineficiente.

Se o número de mensagens se mantiver próximo dos limites, podem ocorrer problemas de início e término rápidos.

Para manter a vazão em níveis aceitáveis \cite{TRAN}\cite{LI}\cite{IMAI} o uso da técnica de heurísticas com abordagem baseada em limites por um determinado tempo se mostra interessante quando aplicada para calcular o número de processos consumidores que serão iniciados \cite{IMAI}. Estes algoritmos podem ser usados para sanar o problema do limiar de detecção de elasticidade para cima ou para baixo. Além disso, a técnica de manter o processo criado por mais tempo para minimizar o custo de reinício de processos pelo sistema operacional diminui a latência de consumo de mensagens.

Por último, em \cite{GUO} é implementada uma abordagem que permite aos desenvolvedores separar a lógica dos programas do controle dos ambientes de computação de forma transparente. Esta é uma característica desejável para reduzir a complexidade das tarefas de gerenciamento que os desenvolvedores precisam lidar no código. Isto também pode ajudar a otimizar o uso de recursos sob restrições de custo para alcançar a qualidade desejada, uma vez que os códigos desenvolvidos serão mais simples de desenvolver e realizar manutenções corretivas e evolutivas. Esta abordagem é interessante em aplicações cujos ambientes possuem diferentes restrições de funcionamento. Quando as restrições de elasticidade são controladas em função de requisitos de qualidade que não necessitam de estratégias diferenciadas, não há a necessidade de uso de diretivas dinâmicas dentro do código dos programas.

\chapter{Solução Proposta}

\chapterprecishere{Este capítulo propõe uma solução de implementação de middleware baseada nas abordagens dos artigos científicos analisadas no capítulo anterior.}

Como apresentado no capítulo anterior, o objetivo deste trabalho é implementar um \emph{middleware} tolerante a falhas que garanta elasticidade para o \emph{cluster} da empresa Sysofit. Para isto, neste trabalho será usada a arquitetura de publicação de novos dados pelos produtores com os dados persistidos pelo coordenador em filas de mensagens \cite{FANG}, após serem gravados em um banco de dados. Esta abordagem será adotada tal que quando um nó falhar não seja afetada a integridade dos demais \cite{WANG}\cite{CASTRO}, e ao ocorrer queda ou travamento de algum processo a operacionalidade possa ser facilmente recuperada e realizada por outro grupo de serviços que acessem os mesmos dados \cite{HE}.

No \emph{middleware} o processo coordenador realizará o papel de \emph{publisher} para os processos que serão os \emph{subscribers}, sendo sinalizados por meio de eventos de existência de itens a serem consumidos nas filas. O suporte às características de tolerância a falhas será implementado na biblioteca de comunicação em nível de \emph{middleware}, e de forma transparente a aplicação \cite{MARTINS}\cite{CASTRO}.

Além disso, será adotada a técnica de replicação de nós para tolerância a falhas com uma implementação \emph{stateless} (sem estado) de baixa dependência e acoplamento entre os processos. Os coordenadores auxiliares serão iniciados rapidamente \cite{ABBES}\cite{BICER} e responderão às requisições dos processos somente em caso de falha ou queda. Os processos, caso detectem uma falha de comunicação com o coordenador, irão esperar até que o novo coordenador inicie e responda às requisições.

Para os processos que possuem estado, será usada a abordagem de \emph{snapshots} periódicos, juntamente com a detecção de falha por meio da queda da conexão TCP entre os processos e o coordenador. Os \emph{snapshots} serão persistidos em um banco de dados para que em caso de detecção de falha algum outro serviço assuma seu processamento recuperando o último \emph{checkpoint} válido, mas sem o envio constante dos estados para os processos replicados para que não ocorra sobrecarga de comunicação.

Para a implementação do \emph{middleware} proposto também será usado IPC (\emph{Inter Process Communication}) com TCP (\emph{Transmission Control Protocol}) para obter entrega confiável (retransmissão, controle de fluxo, fragmentação), ordenação total de pacotes \cite{ZHAO}, deteccão de falhas de nós por meio da queda da conexão \cite{BICER}\cite{MARTINS} e \emph{pings} periódicos com \emph{timeouts} \cite{MARTINS}.

Além disso, a consistência do sistema será feita por meio de \emph{locks} usados para garantir a integridade sequencial, persistindo e roteando os dados que possuem \emph{cache} dentro de regiões críticas no coordenador.

Nesse cenário, também será usada a carga dinâmica de códigos que possuem funcionalidades de um grupo de serviços específico \cite{MA}. Assim, qualquer serviço poderá processar qualquer tipo de mensagem por meio da sua subscrição no coordenador e da carga da \emph{Dll} que possue o código que realizará este processamento. Porções de código serão ativadas para qualquer processo consumidor e estes podem estar rodando em servidores distintos. Quando uma \emph{Dll} for carregada será criada uma nova \emph{worker thread} para processar o novo conjunto de filas associado. Os processos consumidores agruparão porções de código distintas que tratam o mesmo grupo de mensagens para reduzir a latência de início de novos processos, além de representar uma economia no número total de processos necessários para rodar o sistema distribuído.

Assim, suponha que existam cinco funcionalidades distintas para o mesmo tipo de mensagem, e que existam quatro processos genéricos em execução, sendo que três destes estão saturados em consumo de CPU. O coordenador elegerá o consumidor menos sobrecarregado e iniciará uma nova \emph{worker thread} para tratar uma funcionalidade sem a necessidade de subir um novo processo. Isto pode representar também uma economia de recursos, uma vez que iniciar uma nova \emph{thread} é mais leve, pois o sistema operacional não precisa manter as tabelas de controle que consomem memória adicional e nem possui a latência associada ao início de um novo processo. O coordenador realizará a detecção de saturação de CPU de um consumidor e, dinamicamente, realizará a migração de processamento para um processo menos sobrecarregado através da realocação da fila de consumo.

De acordo com a vazão de consumo, as filas de mensagens serão, automaticamente e dinamicamente distribuídas em novas instâncias de filas que serão criadas para escalar quando forem requeridas, ou redistribuídas em um subconjunto menor quando a carga diminuir. O coordenador usará um algoritmo que se adaptará à carga isolando os produtores e consumidores mais importantes, e redirecionando-os para novas instâncias de filas criadas a partir dos mesmos dados com o objetivo de aumentar a vazão. O coordenador será o responsável pelo agrupamento das mensagens em filas correlacionadas e pela atribuição das filas aos processos que realizarão o seu consumo.

Sendo assim, suponha que uma determinada fila possua 9.000 mensagens e que o coordenador detecte a necessidade de ativar três processos para alcançar a vazão necessária para manter a SLA e para que não ocorra enfileiramento residual. Nesta fila se encontram 2.000 mensagens do grupo A, 3.000 mensagens do grupo B e 4.000 mensagens do grupo C. Assim, o coordenador pode utilizar esta abordagem para redistribuir os dados em três filas distintas, uma para cada novo processo, sem perder a integridade sequencial e mantendo o aumento de vazão no consumo.

Para resolver isso será adotada uma abordagem para determinar os melhores candidatos ao consumo de novos itens baseado no menor consumo de CPU médio e na maior vazão média identificada \cite{LI}. O parâmetro que será utilizado pelo \emph{middleware} será o tempo médio de consumo de cada item de dado na fila de mensagens \cite{LEITNER}.

O coordenador, baseado no número de mensagens das filas e no consumo médio de cada mensagem, determinará se a fila será totalmente consumida no intervalo do cronograma de conexão\footnote{Tempo em que o sistema realiza o envio e recebimento de mensagens com os servidores que se comunicam com as unidades móveis instaladas no veículos monitorados pela Sysofit.} e iniciará mais processos, se necessário, para consumir os dados em paralelo e conter o crescimento de mensagens. O número médio de novos itens nas filas será usado como uma abordagem pró-ativa para que o sistema possa se preparar previamente subindo o número de processos necessários para suportar uma demanda de pico prevista baseada em históricos e diminuir a latência de inicialização dos processos que seria feita no modo reativo.

Dessa forma, será usada a técnica de heurística baseada em limites por um determinado tempo \cite{IMAI}, e será aplicado ao calcular o número de processos consumidores que serão iniciados para manter a vazão em níveis aceitáveis. Estes algoritmos serão usados no coordenador para sanar o problema do limiar de detecção de elasticidade para cima ou para baixo. O coordenador também usará a técnica de manter o processo criado por mais tempo para minimizar o custo de reinício de processos pelo sistema operacional e diminuir a latência de consumo de mensagens.

As falhas por inconsistência de \emph{cache} podem ocorrer por não encontrar em memória as informações necessárias para o tratamento da mensagem. Nesse caso, o sistema deve ser tolerante tentando recarregar do banco de dados essas informações. Se mesmo assim o problema persistir, a inconsistência se deve a um erro de configuração e deve ser colocada em um repositório para análise e o sistema deve alarmar a situação. A falha por queda de processo pode ocorrer por situações de concorrência que geram exceções não tratadas. Nesse caso, a mensagem deve ser primeiro tratada por outro processo replicado. Se mesmo assim a queda persistir, a mensagem será colocada em um repositório para análise posterior, pois processar a mesma mensagem em um outro processo replicado gerará o mesmo erro.

\chapter{Cronograma}

Este trabalho será desenvolvido por meio de nove tarefas listadas a seguir, e por meio do cronograma apresentado na Tabela \ref{tab:cronograma}.

1. Levantamento do estado da arte sobre tolerância a falhas e elasticidade para \emph{middleware};

2. Escolha das abordagens que serão utilizadas na implementação das características de tolerância a falhas e elasticidade do \emph{middleware} proposto;

3. Implementação da solução de \emph{middleware} com suporte a tolerância a falha;

4. Implementação da solução de \emph{middleware} com suporte a elasticidade;

5. Testes e análise da solução proposta;

6. Migração e adequação para o ambiente real da arquitetura da Sysofit;

7. Publicação de artigos científicos;

8. Escrita da dissertação de mestrado;

9. Defesa do mestrado.

\begin{table}[!htpb]

\centering

\caption{Cronograma das Atividades deste Trabalho}

\begin{center}

\begin{tabular}{|l|c|c|c|c|c|c|c|c|}

\hline

Mês &Ago/13 &Set/13 &Out/13 &Nov/13 &Dez/13 &Jan/14 &Fev/14 &Mar/14\\

\hline\hline

Tarefa 1 &X &X &X &X & & & & \\

\hline

Tarefa 2 & & &X &X & & & & \\

\hline

Tarefa 3 & & &X &X &X & & & \\

\hline

Tarefa 4 & & &X &X &X & & & \\

\hline

Tarefa 5 & & & &X &X & & & \\

\hline

Tarefa 6 & & & & & &X & & \\

\hline

Tarefa 7 & & & & &X &X &X &X \\

\hline

Tarefa 8 & & & & & &X &X & \\

\hline

Tarefa 9 & & & & & & & &X \\

\hline

\end{tabular}

\label{tab:cronograma}

\end{center}

\end{table}

\chapter{Conclusão}

Este trabalho apresentou conceitos relacionados à sistemas distribuídos, à plataforma \emph{cluster} e à camada de software \emph{middleware}, que é responsável pela abstração de programação das características da arquitetura de computação distribuída utilizada. Além disso, foram apresentados o problema e o cenário do sistema distribuído da Sysofit e analisados diversos artigos científicos recentes para propor uma solução de \emph{middleware} para suportar tolerância a falhas e elasticidade.

Em seguida foi apresentado um cronograma para a realização das atividades apresentadas durante o trabalho.

%Com a utilização do \emph{middleware} proposto espera-se que seja possível otimizar o consumo das filas de mensagens do \emph{cluster}, de forma que novos processos sejam adicionados ou removidos sob demanda ao ocorrer o enfileiramento ou diminuição de mensagens controlados por um limite máximo ou por uma detecção de queda de processos.

%

%Isso contribuirá para que o sistema tenha elasticidade, a configuração de processos seja automática e sob demanda, sem a necessidade de intervenção manual. Além disso, espera-se que o sistema seja tolerante a falhas com alta disponibilidade em caso de queda de processos ou servidores.

%

%Com o suporte à elasticidade e à tolerância a falhas é esperado que ocorra uma diminuição do número de servidores necessários para tratar as mensagens do sistema, liberando recursos computacionais do \emph{cluster} para outras atividades, e diminuindo os tempos de atualização do sistema.

%

%Também é esperado que mesmo em caso de falhas de servidores o sistema seja tolerante e continue a funcionar com alta disponibilidade, sem a necessidade de intervenção manual.

Com o desenvolvimento do \emph{middleware} com suporte a elasticidade espera-se que seja possível economizar recursos de TI por meio da diminuição do número de servidores necessários para processar as mensagens de posicionamento dos veículos monitorados pela Sysofit.

Assim, por meio deste \emph{middleware} espera-se poder dimensionar corretamente o número de servidores necessários para tratar as demandas de processamento de mensagens sem comprometer a qualidade de serviço imposta pelos clientes da Sysofit. Consequentemente, isso poderá reduzir os custos com o hardware necessário para executar as aplicações distribuídas, pois o dimensionamento real, sob demanda, será calculado dinamicamente pelo próprio sistema.

Com a diminuição do número de servidores, o tempo de reinicialização do sistema por conta de manutenções corretivas ou evolutivas para atualizações também poderá ser reduzido. Dessa forma, com o suporte a tolerância a falhas no \emph{middleware} o sistema poderá se tornar resistente a queda ou falha de processos durante o tratamento de mensagens, possibilitando alcançar alta disponibilidade no \emph{cluster}.

% ---

% ---

% Finaliza a parte no bookmark do PDF, para que se inicie o bookmark na raiz

% ---

% ---

% ---

% Conclusão

% \textemdash\

% ----------------------------------------------------------

% ELEMENTOS PÓS-TEXTUAIS

% ----------------------------------------------------------

%\postextual

% ----------------------------------------------------------

% Referências bibliográficas

% ----------------------------------------------------------

%\nocite{\*}

%\bibliographystyle{abntex2-alf}

\bibliography{bib}

% ----------------------------------------------------------

% Glossário

% ----------------------------------------------------------

%

%\glossary

\bookmarksetup{startatroot}%

% ----------------------------------------------------------

% Apêndices

% ----------------------------------------------------------

% ---

% Inicia os apêndices

% ---

% ---

% ----------------------------------------------------------

% Anexos

% ----------------------------------------------------------

% ---

% Inicia os anexos

% ---

\end{document}