UNIVERSIDADE DA REGIÃO DE JOINVILLE – UNIVILLE

DEPARTAMENTO DE INFORMÁTICA

netHPC

UMA IMPLEMENTAÇÃO EM CÓDIGO FONTE ABERTO DE COMPUTAÇÃO DE ALTO DESEMPENHO UTILIZANDO MICROSOFT .NET®

GEORGE LUIZ BITTENCOURT

PROFESSOR DR. ALEXANDRE CIDRAL

Joinville - SC

2009

GEORGE LUIZ BITTENCOURT

netHPC

UMA IMPLEMENTAÇÃO EM CÓDIGO FONTE ABERTO DE COMPUTAÇÃO DE ALTO DESEMPENHO UTILIZANDO MICROSOFT .NET®

Trabalho de Conclusão de Estágio apresentado ao curso de Sistemas de Informação da Universidade da Região de Joinville – UNIVILLE – como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação. Orientador Geral: Prof. MSc. Paulo Marcondes Bousfield. Orientador Específico: Prof. Dr. Alexandre Cidral.

Joinville - SC

2009

FOLHA DE APROVAÇÃO

O acadêmico **GEORGE LUIZ BITTENCOURT**, regularmente matriculado na **4ª SÉRIE** do **CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**, apresentou e defendeu o presente **TRABALHO DE CONCLUSÃO DE ESTÁGIO**, obtendo da banca examinadora a média final \_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_), tendo sido considerado \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ na disciplina.

Joinville, de dezembro de 2009.

**Prof. Dr. Alexandre Cidral**

Orientador Específico e Avaliador

**Prof. MSc Paulo M. Bousfield**

Orientador Geral

**Prof. Convidado**

Avaliador Convidado

**NOME DO ESTAGIÁRIO**: GEORGE LUIZ BITTENCOURT

**UNIVERSIDADE DA REGIÃO DE JOINVILLE – UNIVILLE**

**TRABALHO DE CONCLUSÃO DE ESTÁGIO CURRICULAR SUPERVISIONADO**

**AVALIAÇÃO DO ESTAGIÁRIO PELA EMPRESA**

|  |  |
| --- | --- |
| **QUADRO I – AVALIAÇÃO NOS ASPECTOS PROFISSIONAIS** | |
| **ASPECTO** | **PONTOS** |
| QUALIDADE DO TRABALHO – Considerando o possível | 5 |
| ENGENHOSIDADE – Capacidade de sugerir, projetar, executar modificações ou inovações | 4 |
| CONHECIMENTO – Demonstrado no desenvolvimento das atividades programadas | 5 |
| CUMPRIMENTO DAS TAREFAS – Considerar o volume de atividades dentro do padrão razoável | 4 |
| ESPÍRITO INQUISITIVO – Disposição demonstrada para aprender | 5 |
| INICIATIVA – No desenvolvimento das atividades | 5 |
| **TOTAL I** | 28 |

|  |  |
| --- | --- |
| **QUADRO II – AVALIAÇÃO DOS ASPECTOS HUMANOS** | |
| **ASPECTO** | **PONTOS** |
| ASSIDUIDADE – Cumprimento do horário e ausência de faltas | 5 |
| DISCIPLINA – Observância das normas internas da empresa | 5 |
| SOCIABILIDADE – Facilidade de se integrar com os outros no ambiente de trabalho | 3 |
| COOPERAÇÃO – Disposição para cooperar com os demais para atender as atividades | 5 |
| SENSO DE RESPONSABILIDADE – Zelo pelo material, equipamentos e bens da empresa | 5 |
| **TOTAL II** | 23 |

|  |  |
| --- | --- |
| **AVALIAÇÃO FINAL** | |
| Total do Quadro I multiplicado por 7 | 196 |
| Total do Quadro II multiplicado por 3 | 69 |
| **TOTAL GERAL** | 265 |

**CONCEITO CONFORME TOTAL GERAL**

EXCELENTE

**LIMITES PARA CONCEITUAÇÃO**

De 57 a 101 – SOFRÍVEL

De 102 a 146 – REGULAR

De 147 a 194 – BOM

De 195 a 240 – MUITO BOM

De 241 a 285 - EXCELENTE

**PONTUAÇÃO**:

Sofrível: 1 ponto – Regular: 2 pontos – Bom: 3 pontos – Muito Bom: 4 pontos – Excelente: 5 pontos

Carlos Eduardo Bittencourt Fersura

Joinville - SC, 30 de novembro de 2009.

CARIMBO DA EMPESA

**NOME DA EMPRESA**: Integral Informática Ltda

**REPRESENTADA PELO SUPERVISOR**: Carlos Eduardo Bittencourt Fersura

AGRADECIMENTOS

*Agradeço a força criadora do universo que me concedeu a vida e também a energia para chegar ao término deste trabalho.*

*Agradeço a minha família e em especial a minha esposa Liriane pela atenção, paciência e encorajamento que estes me deram durante todo o período em que me dediquei à conclusão deste trabalho.*

*Aos professores e colegas de classe pelas experiências vividas durante estes últimos anos.*

*A Integral Informática, em especial ao Carlos e ao Sandro, por todo o aprendizado que me proporcionaram ao longo dos tempos.*

*E finalmente, a todos os criadores da ciência da computação, pois criaram algo no qual eu me identifico e me divirto a cada problema resolvido!*

EPÍGRAFE

*Veni, vidi, vici*

*Gaius Julius Caesar, 47 AC*

RESUMO

Este trabalho mostra as atividades executadas pelo acadêmico durante seu estágio no curso de Bacharelado em Sistemas de Informação. Durante esse estágio o acadêmico construiu em caráter experimental um *middleware* de computação em grade a fim de validar os benefícios da computação paralela e distribuída. O trabalho contém a fundamentação teórica utilizada pelo acadêmico e também o detalhamento da parte prática. Ao final do trabalho o acadêmico executou testes onde o cálculo de números primos foi distribuído entre vários processadores, e com o resultado desses testes ficou comprovado cientificamente à aplicabilidade desse tipo de computação em situações onde o quesito tempo de processamento deve ser o menor possível.

**Palavras-chave**: serial; paralelo; computação; distribuída; performance; *middleware*; *grid*; *hpc.*

ABSTRACT

This paper shows the activities executed by the academic during his internship on the Information Systems Bachelor course. During this internship the academic built in a experimental way a grid computing middleware to validate the benefits of the parallel and distributed computing. The paper contains all bibliographic material used by the academic and also the development details. In the end the academic executed a test where the prime numbers calculation was distributed among several processors and with the results the benefits of this model were scientifically proved where the processing time requirement must be the smallest one.

**Keywords**: serial; parallel; computation; distributed; performance; middleware; grid; hpc

LISTA DE FIGURAS

[Figura 1 - Maquete de uma ponte 28](#_Toc248239851)

[Figura 2 - Simulação de um modelo matemático no MATLAB 2009 29](#_Toc248239852)

[Figura 3 - Exemplo de algoritmo 30](#_Toc248239853)

[Figura 4 - Processamento Sequencial 32](#_Toc248239854)

[Figura 5 - Processamento Paralelo 32](#_Toc248239855)

[Figura 6 - Arquitetura da computação em grade 35](#_Toc248239856)

[Figura 7 - Modelo de referência OSI 39](#_Toc248239857)

[Figura 8 - Modelo de Entidades e Relacionamentos 42](#_Toc248239858)

[Figura 9 - Diagrama de casos de uso 48](#_Toc248239859)

[Figura 10 - Diagrama de classes 48](#_Toc248239860)

[Figura 11 - Diagrama de estados 49](#_Toc248239861)

[Figura 12 - Diagrama de domínio do netHPC 54](#_Toc248239862)

[Figura 13 - Diagrama de classes do kit de desenvolvimento do netHPC 56](#_Toc248239863)

[Figura 14 - Diagrama de casos de uso 64](#_Toc248239864)

[Figura 15 - Tela para conexão ao netHPC 66](#_Toc248239865)

[Figura 16 – Tela para inclusão de algoritmo 67](#_Toc248239866)

[Figura 17 - Edição de algoritmo 69](#_Toc248239867)

[Figura 18 - Parametrização da execução 71](#_Toc248239868)

[Figura 19 - Tela específica de parametrização da execução 71](#_Toc248239869)

[Figura 20 - Escolha dos *computing nodes* 72](#_Toc248239870)

[Figura 21 - Tela com o resumo da execução 72](#_Toc248239871)

[Figura 22 - Relatório de Exemplo 75](#_Toc248239872)

[Figura 23 - Diagrama de estados do agente nos computing nodes 79](#_Toc248239873)

[Figura 24 - Diagrama de estados do agente no head node 81](#_Toc248239874)

[Figura 25 - Arquitetura do netHPC 83](#_Toc248239875)

[Figura 26 - Layout do relatório de execução 85](#_Toc248239876)

[Figura 27 - Tela de cadastro de algoritmo 86](#_Toc248239877)

[Figura 28 - Tela utilizada para iniciar uma execução 87](#_Toc248239878)

[Figura 29 - Modelo de entidades e relacionamentos 88](#_Toc248239879)

[Figura 30 - Diagrama utilizado pelo Entity Framework 89](#_Toc248239880)

[Figura 31 - Ambiente de desenvolvimento utilizado 94](#_Toc248239881)

[Figura 32 - Parte dos projetos que compõem o netHPC no Visual Studio 2008 95](#_Toc248239882)

[Figura 33 - Inclusão de console de gerenciamento na MMC 96](#_Toc248239883)

[Figura 34 - Console de gerenciamento criada sobre a MMC 97](#_Toc248239884)

[Figura 35 - Serviço do netHPC criado no Windows 97](#_Toc248239885)

[Figura 36 - Diagrama de classes do algoritmo para cálculo de números primos 99](#_Toc248239886)

[Figura 37 - Método utilizado para inicializar o RangeBreaker 100](#_Toc248239887)

[Figura 38 - Código fonte C# responsável por gerar os work items 101](#_Toc248239888)

[Figura 39 - Algoritmo utilizado para processar um work item 102](#_Toc248239889)

[Figura 40 - Tela com detalhes do autor do algoritmo 102](#_Toc248239890)

[Figura 41 - Tela com os parâmetros utilizados no teste 103](#_Toc248239891)

[Figura 42 - Tela de seleção dos computadores onde o algoritmo será executado 104](#_Toc248239892)

[Figura 43 - Gráfico da duração e trabalho por número de cores 109](#_Toc248239893)

[Figura 44 - Gráfico do número de Work Items por Número de Cores 109](#_Toc248239894)

[Figura 45 - Instalação do netHPC através do Active Directory 111](#_Toc248239895)

[Figura 46 - Configuração do netHPC através do Active Directory 112](#_Toc248239896)

[Figura 47 - Arquivo de ajuda do kit de desenvolvimento de software 113](#_Toc248239897)

LISTA DE QUADROS

[Quadro 1 - Cronograma das atividades 23](#_Toc248239898)

[Quadro 2 - Os 5 maiores computadores do mundo 33](#_Toc248239899)

[Quadro 3 - Requisito funcional REQFUNC-01 52](#_Toc248239900)

[Quadro 4 - Requisito funcional REQFUNC-02 52](#_Toc248239901)

[Quadro 5 - Requisito funcional REQFUNC-03 52](#_Toc248239902)

[Quadro 6 - Requisito funcional REQFUNC-04 53](#_Toc248239903)

[Quadro 7 - Requisito funcional REQFUNC-05 53](#_Toc248239904)

[Quadro 8 - Requisito funcional REQFUNC-06 53](#_Toc248239905)

[Quadro 9 - Regra de negócio RN-01 59](#_Toc248239906)

[Quadro 10 - Regra de negócio RN-02 60](#_Toc248239907)

[Quadro 11 - Ator ATOR-01 62](#_Toc248239908)

[Quadro 12 - Ator ATOR-02 62](#_Toc248239909)

[Quadro 13 - Ator ATOR-03 63](#_Toc248239910)

[Quadro 14 - Caso de uso CSU-01 66](#_Toc248239911)

[Quadro 15 – Caso de uso CSU-02 67](#_Toc248239912)

[Quadro 16 - Caso de uso CSU-03 68](#_Toc248239913)

[Quadro 17 - Caso de uso CSU-04 69](#_Toc248239914)

[Quadro 18 - Caso de uso CSU-05 72](#_Toc248239915)

[Quadro 19 - Caso de uso CSU-06 73](#_Toc248239916)

[Quadro 20 - Caso de uso CSU-07 75](#_Toc248239917)

[Quadro 21 - Caso de uso CSU-08 76](#_Toc248239918)

[Quadro 22 - Caso de uso CSU-09 76](#_Toc248239919)

[Quadro 23 - Caso de uso CSU-10 77](#_Toc248239920)

[Quadro 24 - Requisito não funcional REQNF-01 82](#_Toc248239921)

[Quadro 25 - Requisito não funcional REQNF-02 82](#_Toc248239922)

[Quadro 26 - Requisito não funcional REQNF-03 82](#_Toc248239923)

[Quadro 27 - Tabela Algorithm 90](#_Toc248239924)

[Quadro 28 - Tabela Execution 91](#_Toc248239925)

[Quadro 29 - Tabela Node 91](#_Toc248239926)

[Quadro 30 - Tabela ExecutionNode 92](#_Toc248239927)

[Quadro 31 - Tabela WorkItem 93](#_Toc248239928)

[Quadro 32 - Tabela Event 93](#_Toc248239929)

[Quadro 33 - Configuração do computador CARLOSWIN7 104](#_Toc248239930)

[Quadro 34 - Configuração do computador CLAUDI-VISTA 105](#_Toc248239931)

[Quadro 35 - Configuração do computador FERNANDO-VISTA 105](#_Toc248239932)

[Quadro 36 - Configuração do computador GEORGE 105](#_Toc248239933)

[Quadro 37 - Configuraçao do computador LEO-VISTA 106](#_Toc248239934)

[Quadro 38 - Configuração do computador LUDGERO-PC 106](#_Toc248239935)

[Quadro 39 - Configuração do computador RONALDO-VISTA 106](#_Toc248239936)

[Quadro 40 - Configuração do computador SANDRO-PC 107](#_Toc248239937)

[Quadro 41 - Configuração do computador THALESF-PC 107](#_Toc248239938)

LISTA DE TABELAS

[Tabela 1 - Resultados dos testes de execução 108](#_Toc248239939)

SUMÁRIO

[**INTRODUÇÃO** 15](#_Toc248239940)

[1. **PROJETO DO ESTÁGIO CURRICULAR SUPERVISIONADO** 16](#_Toc248239941)

[1.1. Identificação do Estágio 16](#_Toc248239942)

[1.1.1. Dados de Identificação do Acadêmico 16](#_Toc248239943)

[1.1.2. Dados de Identificação da Empresa 16](#_Toc248239944)

[1.1.3. Dados dos Responsáveis pelo Estágio 16](#_Toc248239945)

[1.2. Tema 17](#_Toc248239946)

[1.3. Assunto 17](#_Toc248239947)

[1.4. Problema 18](#_Toc248239948)

[1.5. Objetivos 19](#_Toc248239949)

[1.5.1. Objetivo Geral 19](#_Toc248239950)

[1.5.2. Objetivos Específicos 19](#_Toc248239951)

[1.6. Justificativa 20](#_Toc248239952)

[1.7. Metodologia 20](#_Toc248239953)

[1.8. Cronograma 22](#_Toc248239954)

[2. **FUNDAMENTAÇÃO TEÓRICA** 24](#_Toc248239955)

[2.1. Ciência 24](#_Toc248239956)

[2.2. Modelo 26](#_Toc248239957)

[2.3. Simulação 28](#_Toc248239958)

[2.4. Algoritmo 30](#_Toc248239959)

[2.5. Computação de Alto Desempenho 31](#_Toc248239960)

[2.5.1. Supercomputadores 33](#_Toc248239961)

[2.5.2. Computação em Grade 34](#_Toc248239962)

[2.6. Processador 37](#_Toc248239963)

[2.7. Redes de Comunicação 38](#_Toc248239964)

[2.8. Banco de Dados 41](#_Toc248239965)

[2.8.1. Entidades 41](#_Toc248239966)

[2.8.2. Atributos 42](#_Toc248239967)

[2.8.3. Modelo de Entidades e Relacionamentos 42](#_Toc248239968)

[2.8.4. Normalização de Dados 43](#_Toc248239969)

[2.8.5. Sistema Gerenciador de Bancos de Dados 43](#_Toc248239970)

[2.9. Engenharia de Software 44](#_Toc248239971)

[2.10. UML – *Unified Modeling Language* 47](#_Toc248239972)

[3. **DESCRIÇÃO DA PRÁTICA** 50](#_Toc248239973)

[3.1. Local do Desenvolvimento 50](#_Toc248239974)

[3.2. Desenvolvimento da Solução 51](#_Toc248239975)

[3.2.1. Requisitos Funcionais 51](#_Toc248239976)

[3.2.2. Análise dos Requisitos 82](#_Toc248239977)

[3.2.3. Projeto do Sistema 84](#_Toc248239978)

[3.2.4. Programação do Sistema 94](#_Toc248239979)

[3.2.5. Teste em Campo do Sistema 98](#_Toc248239980)

[3.2.6. Implantação do Sistema 109](#_Toc248239981)

[3.2.7. Documentação do Sistema 112](#_Toc248239982)

[**CONSIDERAÇÕES FINAIS** 114](#_Toc248239983)

[**REFERÊNCIAS** 117](#_Toc248239984)

[**APÊNDICE A – RELATÓRIOS DAS EXECUÇÕES DE TESTE** 119](#_Toc248239985)

introdução

O trabalho aqui apresentado objetiva descrever as atividades realizadas pelo acadêmico em seu estágio na Integral Informática Ltda, empresa na qual foi criada uma solução de computação em grade em caráter experimental.

Esse trabalho está divido em três capítulos, nomeados respectivamente como projeto do estágio curricular supervisionado, fundamentação teórica e descrição da prática.

No primeiro capítulo o projeto apresentado inicialmente à disciplina de estágio curricular supervisionado (ECS) é detalhado. Esse capítulo mostra, em uma visão sintética, o que o acadêmico se propôs a desenvolver durante seu estágio supervisionado.

Já o segundo capítulo mostra a fundamentação teórica sobre a qual a parte prática foi concebida. São fundamentados conceitos como ciência, algoritmo, modelo e computação em grade.

Para finalizar, o terceiro capítulo mostra o detalhamento da prática. Nesse capítulo são mostrados os artefatos gerados, a arquitetura do sistema e o resultado dos testes em campo produzidos pelo acadêmico.

# PROJETO DO ESTÁGIO CURRICULAR SUPERVISIONADO

Este capítulo apresenta a definição do projeto executado, mostrando o tema e assunto selecionado para execução no estágio curricular, os objetivos do projeto, as justificativas em relação à necessidade da execução do projeto e as metodologias planejadas para realização das atividades no período de estágio curricular.

## Identificação do Estágio

Na sequência podem ser visualizadas as informações de contato do acadêmico, da empresa onde o estágio foi efetuado e também dos responsáveis pelo estágio.

### Dados de Identificação do Acadêmico

Nome: George Luiz Bittencourt

Curso: Bacharelado em Sistemas de Informação

Endereço: Rua Otto Benack, 321 – Apto 402 – Bom Retiro – Joinville – SC

### Dados de Identificação da Empresa

Denominação: Integral Informática Ltda

Ramo de Atividade: Consultoria em Tecnologia da Informação

Endereço: Rua do Príncipe, 777 – Sala 5 – Centro – Joinville – SC

Fone: (47) 3032-8100 e-mail: falecom@integral.inf.br

### Dados dos Responsáveis pelo Estágio

Orientador de Classe: Prof. MSc. Paulo Marcondes Bousfield

Orientador Específico: Prof. Dr. Alexandre Cidral

Supervisor no Campo de Estágio: Carlos Eduardo Bittencourt Fersura

## Tema

Uma implementação em código fonte aberto de computação de alto desempenho (*HPC*) utilizando Microsoft .NET C# ® e outras tecnologias da Microsoft.

## Assunto

A computação de alto desempenho é um termo antigo nos meios acadêmicos e militares devido ao grande volume de dados analisados por estes setores. Basicamente, uma solução de computação de alto desempenho envolve os seguintes componentes:

1. Um nó central (*head node*) que coordena os demais nós (*computing nodes*) para que o trabalho seja distribuído e processado;
2. Vários nós de computação (*computing nodes*)que efetuam o processamento e se comunicam entre si e com o *head node* a fim de processar os dados;
3. Uma lógica construída de tal maneira que pode ser distribuída entre vários computadores, evitando assim que os mesmos dados sejam processados por mais de um computador. Essa lógica deverá ser construída de acordo com uma interface definida pela solução de *HPC*, permitindo com isso usufruir da infra-estrutura criada pela solução.

O processo de utilização desse tipo de solução basicamente se consiste nos seguintes passos:

1. Um analista com conhecimento dos problemas que se deseja analisar cria um algoritmo especificando os passos que devem ser executados. Exemplo: um engenheiro ambiental cria um algoritmo que tem por objetivo mapear a relação entre as chuvas e a erosão nas margens dos rios;
2. Um desenvolvedor cria um programa otimizado e que pode ser distribuído com base no algoritmo criado pelo analista, utilizando a interface provida pela solução de *HPC* escolhida;
3. O responsável pela solução carrega esse programa e os dados no sistema de *HPC* através do *head node* e inicia o processamento. Não necessariamente o programa precisa que dados externos sejam carregados, estes dados podem ser gerados em tempo de execução pelo programa em cada *computing node*;
4. Cada *computing node* descarrega o programa e inicia a execução. O programa busca os itens de trabalho (se necessário), processa-os e envia o resultado para a *head node* ou para outro meio de armazenamento. É importante frisar que esse programa é um programa normal e pode utilizar qualquer uma das funcionalidades que se usaria em algum outro sistema, ou seja, não existe nenhuma restrição em relação ao que o programa pode fazer;
5. O analista verifica os dados gerados e se necessário efetua ajustes no algoritmo. Caso não seja necessário efetuar nenhum ajuste, o resultado da análise já se encontra processado e pode ser utilizado.

## Problema

Vive-se em um mundo globalizado e digital, onde o número de informações geradas pelas pessoas e empresas cresce vertiginosamente a cada dia. Algumas dessas informações são grandes massas de dados que precisam ser analisadas a fim de se obter um resultado. Utilizando os meios tradicionais, onde somente um computador é empregado, o tempo de processamento é grande. Pensando nesse problema, soluções de *grid computing* e *HPC* nasceram nos meios acadêmicos e militares para distribuir o processamento entre diversos computadores ou em um super computador com diversos processadores.

Com esse tipo de solução surgiu um novo problema, o desenvolvedor não mais podia pensar em execução sequencial, mas sim em execução paralela, onde aquela lógica estaria sendo executada em diversos processadores/computadores ao mesmo tempo, sendo necessário se comunicar e coordenar as unidades de trabalho entre todas as unidades de processamento.

## Objetivos

Na sequência o objetivo geral e os objetivos específicos do estágio são apresentados.

### Objetivo Geral

Desenvolver um produto em caráter experimental e de código fonte aberto, utilizando-se de tecnologias da Microsoft que irá possibilitar a distribuição do processamento em diversos computadores e também fornecer um modelo de programação orientado a objetos para o desenvolvimento da lógica que será distribuída pela solução.

### Objetivos Específicos

Os objetivos específicos definidos para o trabalho de estágio são:

1. Criar um aplicativo central (console de gerenciamento) de onde o administrador poderá efetuar as seguintes tarefas:
   1. Gerenciar os nós de computação. As operações possíveis serão iniciar e interromper a execução de uma lógica distribuída e também atualizar o cliente do netHPC;
   2. Efetuar a manutenção das lógicas distribuíveis. Essa manutenção é a inclusão, exclusão e alteração das lógicas contidas na solução;
   3. Visualizar o resultado das execuções através dos dados enviados pelos *computing nodes*;
2. Desenvolver um segundo aplicativo que será instalado em cada um dos nós de computação e que se comunicará com o aplicativo central através de conexões de rede TCP/IP;
3. Criar um kit de desenvolvimento de software (SDK, software development kit) orientado a objetos, que o desenvolvedor de lógicas distribuíveis deverá utilizar para usufruir das funcionalidades do netHPC. Adicionalmente, este kit possuirá dois algoritmos de exemplo para tornar menor a curva de aprendizado da solução;
4. Construir um programa para efetuar a instalação da solução, tanto no *head node,* quanto nos *computing nodes*;
5. Documentar o sistema em formato de ajuda do Windows a fim de tornar mais fácil sua implantação;
6. Publicar todos os artefatos gerados (códigos fonte, diagramas, etc) no *web site* www.nethpc.com, sob uma licença de código fonte aberto contribuindo assim com a comunidade.

## Justificativa

Com a necessidade de processamento de uma quantidade cada vez maior de dados, sistemas de computação de alto desempenho ganham cada vez mais destaque nas empresas e centros de pesquisas. Seja para mapear o DNA humano, para analisar cenários meteorológicos ou ainda para descobrir vida em outros planetas (projeto SETI).

Muitas empresas possuem grandes massas de dados e podem se beneficiar de um grupo de computadores para efetuar o processamento paralelo destes dados. Atualmente, existem soluções de código fonte aberto e comerciais para esse tipo de problema, porém nenhuma fornece um modelo orientado a objetos em .NET de fácil utilização. É essa forma fácil de desenvolver, modelar e administrar, esse tipo de problema que o netHPC tem por objetivo prover.

Além da falta de uma solução de computação de alto desempenho em .NET, esse tema foi escolhido pelo acadêmico pois é tópico de seu interesse pessoal e além disso, várias tecnologias da Microsoft serão utilizadas no produto, enriquecendo assim o seu conhecimento.

## Metodologia

Para o desenvolvimento do netHPC as seguintes fases serão executadas:

1. Levantamento de Requisitos: nesta fase os requisitos a serem atendidos pela solução serão enumerados. Isso será efetuado através da leitura de livros sobre *HPC* e *grid computing* e também com uma análise das soluções já existentes no mercado, como o **Microsoft** **Windows HPC Server 2008 ®**;
2. Análise dos Requisitos: após os requisitos serem elencados durante a fase de levantamento, os casos de uso e modelos serão gerados. Nessa fase os casos de uso serão priorizados para desenvolvimento;
3. Design dos Requisitos: nesta fase a forma como os requisitos serão implementados será definida. Serão utilizados somente componentes de distribuição gratuita da Microsoft, como: C# ®, MMC ®, SQL Server Compact Edition ®;
4. Construção: essa é a fase do projeto onde a maior parte do tempo será despendida. Os casos de uso priorizados anteriormente serão implementados, gerando o produto em si;
5. Testes e implantação: durante a fase de desenvolvimento os testes serão efetuados em máquinas virtuais, porém, conforme o produto for tornando-se mais estável, testes serão conduzidos em campo utilizando os computadores da Integral para distribuir o processamento.

Toda a especificação do sistema será efetuada utilizando a linguagem de modelagem **UML 2.0** (*unified modelling language*) e também o modelo de entidades e relacionamentos (MER). Dentre os diagramas existentes na linguagem UML, os seguintes serão utilizados:

1. Diagrama de Classes;
2. Diagrama de Caso de Uso;
3. Diagrama de Estados;

## Cronograma

O cronograma do mostra o tempo estimado e realizado de cada atividade do estágio.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Atividade** | **Mai** | | | | **Jun** | | | | **Jul** | | | | | **Ago** | | | | **Set** | | | | **Out** | | | | | **Nov** | | | |
| **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **5** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **5** | **1** | **2** | **3** | **4** |
| Analisar as ferramentas de HPC do mercado |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Efetuar a leitura das referências bibliográficas |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Levantar os requisitos que o netHPC deve atender |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Analisar os requisitos moldando a solução a ser construída |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Efetuar o design do sistema |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Construir a console de gerenciamento do *head node* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Construir o cliente que irá processar os programas através dos *computings nodes* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Efetuar a documentação do sistema no formato de ajuda do Windows® |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Construir o instalador do netHPC |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Construir dois algoritmos de exemplo utilizando a infraestrutura do netHPC |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Efetuar testes em ambiente real |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Efetuar testes durante o desenvolvimento |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Documentar o sistema para o TCE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Quadro - Cronograma das atividades

Fonte: George Luiz Bittencourt (2009)

**Legenda**

|  |  |
| --- | --- |
| Tempo previsto em semanas |  |
| Tempo real em semanas |  |

# FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais nos quais o projeto de estágio foi baseado. São abordados temas relacionados à ciência, método científico, modelos, simulação e principalmente à computação de alto desempenho, que é o cerne do trabalho. O capítulo é encerrado fundamentando os conceitos básicos do desenvolvimento de sistemas, como banco de dados, UML, processo de desenvolvimento de software, entre outros.

## Ciência

Ciência é uma palavra com origem no latim, que significa **conhecimento**. O principal objetivo da ciência, com a geração de conhecimento, é descrever e entender sistemas complexos e também resolver os problemas que afligem a humanidade. Esses sistemas complexos são os átomos, a sociedade humana, o sistema solar, entre outros. Já os problemas que a ciência tenta resolver vão desde a cura de doenças, previsão e controle do clima até formas de se habitar outros planetas.

Esse conhecimento das mais diversas áreas é obtido através de uma forma sistemática, ou seja, existe um método que deve ser seguido a fim de se obter um conhecimento válido e com qualidade para a ciência. Esse método é conhecido como **método científico**.

O método científico foi descrito por Wolfs (2005, web) como:

The scientific method is the process by which scientists, collectively and over time, endeavor to construct an accurate (that is, reliable, consistent and non-arbitrary) representation of the world.

Através do método científico é possível gerar um conhecimento com qualidade, reproduzível, livre de eventuais problemas e que pode ser aplicado por outros cientistas a fim de resolver problemas ou ainda estender o conhecimento adquirido. Ainda segundo Wolfs (2005, web) o método científico é constituído basicamente dos seguintes passos:

1. Observar: neste passo o fenômeno que se deseja estudar é observado e descrito, a fim de obter o maior número possível de dados para posterior análise. Um exemplo dessa observação são os cientistas que saem em campo à procura de tornados, com o objetivo de coletar dados;
2. Formular: no segundo passo, hipóteses são geradas para explicar o comportamento do fenômeno. Para gerar essas hipóteses, **modelos** podem ser criados e executados em computadores com o objetivo de **simular** o comportamento do fenômeno. Seguindo com o exemplo dos tornados, um **algoritmo** poderia ser criado **modelando** um tornado, tornando assim a análise mais eficaz e segura;
3. Validar: consiste em utilizar as hipóteses e modelos gerados no passo anterior para predizer o comportamento do fenômeno sob diferentes variáveis. No exemplo do tornado, uma vez a hipótese gerada e o algoritmo criado, seria possível alterar as variáveis do meio ambiente como: pressão barométrica, velocidade do vento, e com isso, **simular** o tornado sob essas condições;
4. Avaliar: o último passo, e um dos mais importantes, consiste em outros cientistas avaliarem as hipóteses e observações geradas. Esse passo é fundamental, pois erros nas hipóteses podem ser descobertos, aumentando com isso a qualidade do conhecimento gerado. Essa avaliação geralmente é efetuada através da publicação dos resultados em artigos em revistas científicas, Internet, conferências, etc.

Como pode ser visto, o método científico é vital para a obtenção do conhecimento. Alguns exemplos de problemas estudados pela ciência e que utilizam o método científico, e também necessitam de processamento de alto desempenho devido ao grande volume de dados existentes, podem ser visualizados na sequência. Para cada um desses problemas é possível gerar um algoritmo e utilizar computação de alto desempenho na auxiliar no processo de análise.

* Mapeamento do DNA humano através do projeto GENOMA. Esse projeto tem por objetivo identificar todos os 100.000 genes no DNA humano e mapear as relações entre estes. Com o conhecimento adquirido através desse projeto será possível entender melhor a genética humana e assim tratar melhor as doenças que afligem o ser humano;
* Identificar possíveis planetas que contenham vida inteligente através do projeto SETI (*Search for Extraterrestrial Intelligence*). Esse projeto captura frequências de rádio vindas do espaço e as analisa em busca de padrões de comunicação. Com isso, em teoria, é possível determinar possíveis planetas com vida;
* Análises climáticas a fim de efetuar previsões de chuvas, secas, tsunamis, e outras intempéries que podem acontecer, minimizando assim o risco causado aos seres humanos.

A ciência, através dos centros de pesquisas e universidades, é o maior cliente de soluções de computação de alta performance, visto as várias análises e dados que essa precisa processar para gerar conhecimento.

O trabalho aqui apresentado é de cunho experimental, que conforme Neto (2002) é um estudo no qual se alteram os valores das variáveis de controle do fenômeno ou objeto sendo estudado, com o objetivo de verificar o impacto que essas alterações provocam nestes. Define ainda Neto (2002) que a variável de manipulação é aquela que se acredita que influenciará ou ainda afetará o fenômeno ou objeto sendo estudado, essa variável é ainda chamada de variável independente. As demais variáveis estudadas são chamadas de dependentes, pois as alterações nas variáveis independentes causam impactos nessas.

No contexto do tema do trabalho, a variável de manipulação é o número e marca dos processadores utilizados no processamento dos algoritmos desenvolvidos. Já a variável de observação é a duração do processamento desses algoritmos.

## Modelo

Rumbaugh (2006) afirma que um modelo é uma representação de um fenômeno ou objeto, construído com o objetivo de entender antes de construir. Os modelos geralmente omitem detalhes não essenciais e devido a isso tornam a manipulação mais fácil do que a entidade ou evento real que está sendo representada. Modelos permitem ainda lidar com fenômenos ou objetos complexos e difíceis de simular de forma mais fácil e controlável.

Ainda segundo Rumbaugh (2006), modelos possuem as seguintes finalidades:

1. Testar antes de construir: permitem **simular** situações antes de construir o objeto real, com isso os custos de um eventual problema de projeto são reduzidos. No caso de um prédio é muito melhor descobrir algum problema estrutural em um modelo do que no prédio real;
2. Comunicar: modelos podem ser utilizados para apresentar idéias para clientes, cientistas ou outras pessoas envolvidas. É mais fácil transportar um modelo para alguém avaliar do que a objeto real;
3. Visualizar: permitem visualizar uma idéia antes de construir qualquer objeto. Como exemplo temos os diagramas UML, que permitem visualizar os detalhes de um sistema antes de construí-lo;
4. Reduzir a complexidade: é provavelmente essa a principal finalidade dos modelos, pois permitem lidar com sistemas complexos de forma mais fácil.

Alguns exemplos de modelos utilizados:

1. Maquetes: representam um objeto físico como um prédio, avião ou ponte. A mostra uma ponte que se deseja construir em forma de maquete;
2. Fórmulas Matemáticas: são modelos matemáticos que representam a idéia de causa e efeito de um fenômeno, onde as variáveis independentes causam alterações na variável dependente. Um exemplo de um modelo matemático são as derivadas que tem por objetivo medir a variação de uma função;
3. Diagramas UML: mostram as diferentes perspectivas de um sistema, como por exemplo, o diagrama de classes mostra as entidades e relacionamentos de um sistema e o diagrama de casos de uso mostra as interações dos atores com o sistema;
4. Modelos Climáticos: modelam cenários climáticos, como tempestades, tsunamis ou furacões. São muito utilizados atualmente para entender as alterações climáticas pelo qual o planeta está passando. Esses modelos são criados através de **algoritmos** que são processados por sua vez em computadores.

Em soluções de computação de alto desempenho esses modelos são representados através de **algoritmos**, que contém a lógica utilizada para modelar o objeto ou fenômeno sendo estudado.



Figura - Maquete de uma ponte

Fonte: www.bridgepros.com, acesso em 15/08/2009

## Simulação

Conforme Law (1999), simular é imitar as operações e comportamentos de algo. A simulação provê os seguintes benefícios:

1. Reproduzível: uma vez criado o modelo, várias simulações podem ser efetuadas com esse a qualquer momento;
2. Seguro: a simulação permite um acompanhamento em um ambiente seguro e controlado, eliminando assim o risco à vida das pessoas envolvidas;
3. Baixo Custo: é mais barato efetuar simulações em um modelo do que no objeto real.

Utilizando os exemplos citados na definição de modelo, as seguintes simulações podem ser efetuadas com estes:

1. Maquetes: as maquetes podem ser testadas em túneis de vento a fim de verificar o comportamento em fortes tempestades, a resistência dos materiais pode ser verificada, entre outros;
2. Fórmulas Matemáticas: fórmulas podem ser validadas, comportamentos de fenômenos podem ser conhecidos, entre outros. A mostra um modelo matemático sendo simulado na ferramenta matemática MATLAB;
3. Diagramas UML: utilização de sistemas podem ser simuladas com diagramas, esboços de telas e outros artefatos;
4. Modelos Climáticos: tempestades, tsunamis, e outros podem ser criados em computador a qualquer momento.

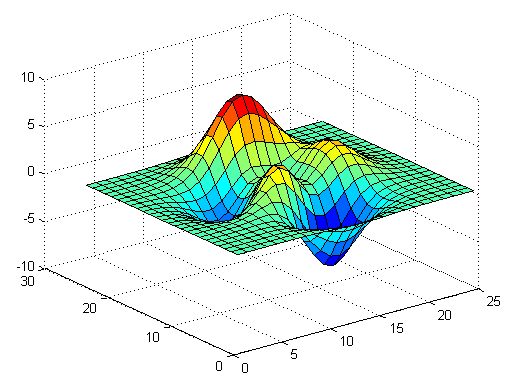


Figura - Simulação de um modelo matemático no MATLAB 2009

Fonte: Arquivo de ajuda do MATLAB R2009a

Na computação de alto desempenho a simulação se apresenta como uma **execução** de um algoritmo. Essa execução pode ser parametrizada com o objetivo de alterar o comportamento do modelo representado pelo algoritmo.

## Algoritmo

Segundo Cormen(2001, p. 10):

Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship.

Um algoritmo, do ponto de vista da ciência da computação, nada mais é que uma série de passos executados de forma sequencial contendo instruções condicionais, repetições e atribuições. É possível comparar um algoritmo com uma função matemática, pois ambos possuem entradas, saídas e um protocolo que deve ser seguido para gerar as saídas.

Um algoritmo pode, inicialmente, ser definido em formato de pseudocódigo conforme mostra a . Nessa figura é possível visualizar um algoritmo de validação de frequências, que poderia muito bem ser utilizado no projeto SETI já citado anteriormente. Basicamente o algoritmo itera sobre um conjunto de frequências disponíveis verificando se a frequência se encontra dentro dos limites audíveis pelo ser humano. Quando uma é encontrada uma mensagem é impressa na tela.

|  |
| --- |
| **repita** **para cada** amostra de freqüência  **se** freqüência maior ou igual a 20 hertz  **se** freqüência menor ou igual a 20 kilohertz  **imprimir** “Possível planeta habitado encontrado”  **fim se**  **fim se**  **fim do repita** |

Figura - Exemplo de algoritmo

Fonte: George Luiz Bittencourt (2009)

Quando o algoritmo for definido em pseudocódigo é necessário traduzi-lo para uma linguagem de programação e essa por sua vez deve ser compilada para um código executável pelo processador. Esse processo de tradução é executado por um compilador, que é uma ferramenta que conhece tanto a sintaxe e semântica das linguagens de programação de origem e destino e também as especificidades do ambiente (sistema operacional, processador), onde o código será executado.

## Computação de Alto Desempenho

A computação de alto desempenho (*high performance computing*, sigla: HPC) trata do processamento de uma grande quantidade de dados de forma **paralela**, conseguindo com isso diminuir consideravelmente o tempo necessário de processamento em relação ao processamento de forma **sequencial**.

Segundo Culler et al (1998, p. 83) para utilizar o processamento paralelo são necessários os seguintes passos:

1. Decompor: é a fase onde o trabalho total é decomposto literalmente em unidades menores;
2. Atribuir: é o processo de atribuir uma unidade de trabalho menor para uma unidade de processamento;
3. Orquestrar: consiste em gerenciar a decomposição e distribuição do processamento entre as unidades de processamento, prover a comunicação e controlar o início e fim das execuções;

Os passos de **decompor** e **atribuir** são conhecidos também como **particionamento**, pois particionam o trabalho total em fatias. Grama et al (2003, p. 31) define esse particionamento como “*dividing a computation into smaller computations and assigning them to different processors for parallel execution are the two key steps in the design of parallel algorithms*”, ou seja, dividir o trabalho total e atribuir para as unidades de processamento as unidades de trabalho é a principal tarefa de uma solução de computação de alto desempenho..

Ao analisar os conceitos introduzidos por Culler et al (1998) fica claro que para processar dados de forma paralela implica em haver mais de uma **unidade de processamento**. Essas unidades de processamento são os processadores que existem dentro de cada computador. O segmento de processadores se encontra na era da programação paralela, pois as fabricantes de processadores como Intel e AMD estão aumentando capacidade de processamento através do aumento da quantidade de processadores por encapsulamento físico. Com isso um único computador pode facilmente contar com vários processadores para utilização.

Na é possível visualizar o fluxo de processamento no modo sequencial. Nesse modo existe somente uma unidade de processamento e essa processa **uma** única unidade de trabalho por vez. Nesse modo o tempo total de processamento é igual à soma do tempo de processamento de cada unidade de trabalho.

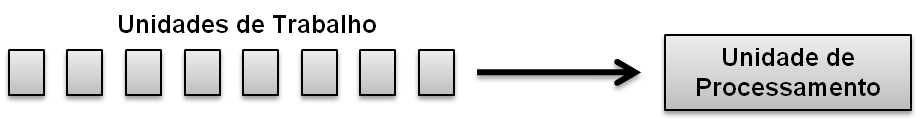


Figura - Processamento Sequencial

Fonte: George Luiz Bittencourt (2009)

Já na existem várias unidades de processamento e cada uma processa uma unidade por vez em paralelo. Com isso o tempo total de processamento está relacionado ao número de unidades de processamento disponíveis e também ao tempo para processar cada unidade de trabalho.

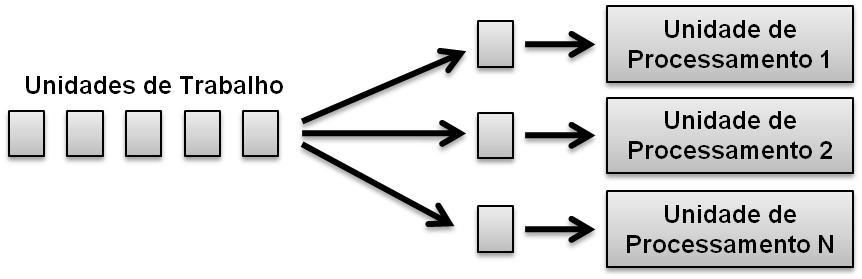


Figura - Processamento Paralelo

Fonte: George Luiz Bittencourt (2009)

Ambos os modos de processamento chegarão ao mesmo resultado, mas quando existe uma grande quantidade de dados e o tempo disponível para obter o resultado é curto, o modo paralelo é o mais indicado, pois divide o trabalho total entre as diversas unidades de processamento.

Apesar da computação de alto desempenho ser mais empregada nos meios científicos e militares, ela também pode ser utilizada no mundo dos negócios. Simulações econômicas podem utilizar a computação de alto desempenho a fim de verificar o resultado de planos econômicos, problemas em projetos de engenharia podem ser simulados, entre outros.

Tendo como base o processamento paralelo e a utilização de várias unidades de processamento, existem atualmente duas abordagens que podem ser utilizadas para atingir a computação de alto desempenho, que são os **supercomputadores** e a **computação em grade**.

### Supercomputadores

Segundo o site GridCafe (2009), supercomputadores são gigantescos (em termos de quantidade de processadores) computadores concentrados em um único lugar físico. Esses supercomputadores contam com centenas de milhares de processadores e custam centenas de milhões de dólares. Esses supercomputadores são utilizados principalmente por grandes órgãos de pesquisa governamentais e pelos militares.

Na é possível visualizar os cincos maiores computadores em termos de poder de processamento. Esses dados são consolidados no web site **TOP 500 Supercomputers** que contém a lista mundial de supercomputadores conhecidos.

|  |  |  |  |
| --- | --- | --- | --- |
| Rank | País | Local | TFlops |
| 1 | Estados Unidos | DOE/NNSA/LANL | 1105.00 |
| 2 | Estados Unidos | Oak Ridge National Laboratory | 1059.00 |
| 3 | Alemanha | Forschungszentrum Juelich (FZJ) | 825.50 |
| 4 | Estados Unidos | NASA/Ames Research Center/NAS | 487.01 |
| 5 | Estados Unidos | Oak Ridge National Laboratory | 478.20 |

Quadro - Os 5 maiores computadores do mundo

Fonte: www.top500.org, acesso em 12/08/2009

Para comparar o desempenho de supercomputadores é utilizada a unidade de medida **flops**. Essa unidade mede a quantidade de operações com ponto flutuante que o processador consegue executar por segundo.

### Computação em Grade

Ainda segundo o site GridCafe (2009), computação em grade (*grid computing* ou ainda *cluster computing*) é uma outra forma de atingir um alto desempenho computacional e é o tema desse trabalho de conclusão.

Na computação em grade não é utilizado um supercomputador, mas sim vários computadores, também conhecidos como nós, interligados e coordenados a partir de um ponto central. A idéia é utilizar vários computadores baratos e no momento que estão ociosos para processar as unidades de trabalho.

Se analisarmos quantos computadores ficam ociosos no período noturno nas empresas ou nas universidades, veremos que existe um grande poder de processamento disponível. Indo mais além, com a Internet todos os computadores do mundo podem ser interligados formando o que é conhecido no meio da computação em grade como **“the grid”**.

#### Arquitetura

Segundo GridCafe (2009):

A grid's architecture is often described in terms of "layers", where each layer has a specific function. The higher layers are generally user-centric, whereas lower layers are more hardware-centric, focused on computers and networks.

A mostra a arquitetura de uma solução de computação em grade.

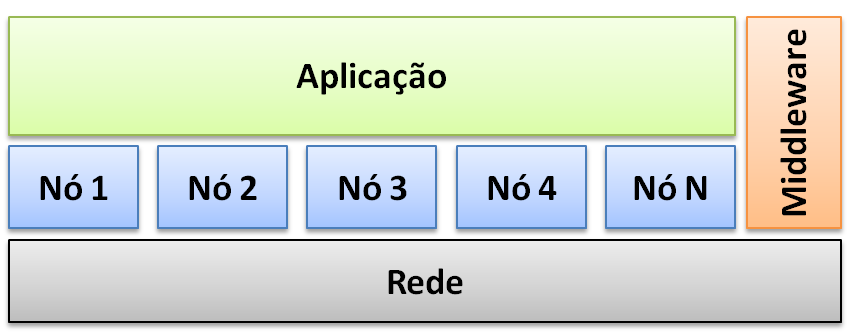


Figura - Arquitetura da computação em grade

Fonte: George Luiz Bittencourt (2009)

Os detalhes de cada camada da arquitetura são explicados a seguir.

#### Rede e Nós

Esses componentes formam a camada física da solução. A rede é o meio de comunicação pelo qual os nós de computação se comunicam e ainda acessam outros recursos como *storages*, sensores, telescópios, e outros. Dependendo da escala do *grid*, o tipo de rede pode ser local onde as velocidades de comunicação são na casa de *gigabits*, ou em *grids* de escala mundial a velocidade tende a ser menor já que redes de longa distância são utilizadas.

Já os nós de computação provêem as unidades de processamento que serão utilizadas no *grid*. Esses computadores possuem um software chamado de agente, que possibilita fazer parte do *grid* e é através desse agente que as unidades de trabalho e aplicações são recebidas para processamento.

#### Middleware

Segundo Foster et al (2004, p. 259), o objetivo do *middleware* é efetuar o gerenciamento dos recursos, que é “*resource management refers to the operations to control how capabilities provided by Grid resources and services are made available to other entities, whether users, applications or services*”. Com base nisso, o *middleware* é o software que controla a distribuição da aplicação para os nós, a divisão e envio das unidades de trabalho e o retorno do processamento.

O *middleware* provê uma interface para administrar os recursos e também uma API que a aplicação pode utilizar para se comunicar com os demais nós, buscar unidades de trabalho para processamento e retornar o resultado do processamento.

#### Aplicação

A aplicação é a lógica (também conhecida como **algoritmo**) que se deseja distribuir com o objetivo de **simular** um **modelo** ou processaruma grande quantidade de dados. O algoritmo por trás desse modelo deve processar as unidades de trabalho, gerando as saídas para análise pelo especialista.

Como esse algoritmo nada mais é que um programa, é importante que o especialista no domínio do problema sendo analisado valide os resultados gerados a fim eliminar eventuais erros de programação existentes.

Esse algoritmo pode utilizar a API provida pelo *middleware* para se comunicar com os demais nós ou pode ainda ser totalmente estanque não se comunicando com os demais computadores.

#### Performance

Como o objetivo da computação distribuída é aumentar a performance do processamento de grandes quantidades de dados, se faz necessário haver meios de mensurar o ganho dessa.

Segundo Breshears (2009, p. 66), em relação ao desempenho da execução paralela:

The time taken to execute is your paramount concern in developing parallel solutions. When doing comparisons, elapsed time is always the final judge of whether the concurrent code is better than the serial one.

Essa afirmação mostra que a medida básica para comparar a execução serial contra a execução paralela é o tempo de cada execução, afinal se o tempo de processamento paralelo for maior que o serial não há uma justificativa para o incremento da complexidade da solução.

Além disso, Breshears ainda introduz a unidade de medida *speedup* para execuções paralelas. Segundo ele (2009, p. 66), speedup é “*the ratio of serial execution time to parallel execution time*”, ou seja, é o resultado da divisão entre o tempo do algoritmo sendo executado serialmente e o tempo em paralelo. Com isso, é possível dizer que em paralelo determinado algoritmo é duas ou três vezes mais rápido que em forma serial.

Em adição a unidade de medida *speedup* existe ainda o indicador da eficiência, que segundo Breshears (2009, p. 70) é

Whereas speedup gives us a metric to determine how much faster our parallel applications are versus their serial brothers, efficiency tells us how well we are utilizing the computational resources of the system

Para calcular a eficiência da execução paralela, divide-se o *speedup* da execução pelo número de *cores* utilizados durante essa. Com isso é possível dizer o quão ocioso os *cores* estavam durante a execução.

É importante frisar que não importa a quantidade de *cores* que se utilize, irá chegar um ponto onde não será mais possível dividir o trabalho em unidades menores e o tempo para processar essa unidade será o tempo mínimo da execução. A duração do processamento não é uma função linear, afinal, se fosse uma, em algum momento a mesma atingiria o valor zero, e quando se trata de processamento, por menor que o tempo seja, ele ainda não será o zero absoluto.

## Processador

Shanley (2004, p. 10) diz que “*the processor’s role in the system is really quite simple: it is an engine designed to fetch instructions from memory, decode them, and execute them*”, ou seja, o processador é um circuito eletrônico presente em todos os computadores, e que quando energizado busca instruções na memória do computador, decodifica e as executa. Ele fica nesse ciclo enquanto estiver energizado ou até que uma instrução de desligamento seja executada.

Shanley (2004) ainda detalhe esse ciclo de execução:

* Fetch: é o ato de buscar uma instrução a partir da memória do computador. Essa memória pode ser tanto volátil (RAM) ou não-volátil (ROM). Essa instrução nada mais é um código que representa a operação a ser executada e também os parâmetros da operação;
* Decode: consiste em decodificar a instrução buscada na memória gerando uma nova instrução entendida internamente pelo processador;
* Execute: é a fase final do ciclo de processamento. Nessa fase a instrução já decodificada é propriamente executada pelo processador alterando o seu estado. Após a execução da instrução o ciclo reinicia buscando a próxima instrução.

Todo processador possui um *instruction set*, que é o conjunto de instruções suportadas pelo processador e disponíveis para utilização pelos sistemas. É tarefa do compilador traduzir o código escrito em alto nível para essas instruções para execução posterior. O conjunto de instruções disponíveis varia de acordo com o modelo e o fabricante do processador.

Ainda segundo Shanley (2004), internamente o processador possui uma série de registradores, que são bancos de memória interna que operam na mesma velocidade do processador. Esses registradores são utilizados a todo momento, porém sua capacidade é limitada. Devido a isso o processador acessa a memória RAM externa a ele, e que possui uma grande capacidade, se comparado aos registradores. Esse acesso não é efetuado na mesma velocidade do processador, mas sim em uma velocidade mais baixa. Devido a isso o compilador tem uma grande tarefa de tentar otimizar ao máximo o código para que esse seja o mais performático possível.

## Redes de Comunicação

Uma rede de comunicação é composta de uma infraestrutura física utilizada para transmitir os *bits* que compõem a mensagem a ser enviada. Os benefícios das redes de comunicação mudaram radicalmente a forma de se comunicar, afinal, hoje com a Internet não existem mais barreiras para a comunicação.

Uma rede de comunicação é construída utilizando o modelo de referência OSI (*Open Systems Interconnection*), que segundo Siyan et al (2002) é “*a model established the global standard for defining the functional layers required for open communications between computers*”. Esse modelo define sete camadas, onde as camadas inferiores transportam os dados das camadas superiores e cada camada tem um objetivo distinto no processo de comunicação.

Na épossível verificar essas camadas e sua respectiva numeração.

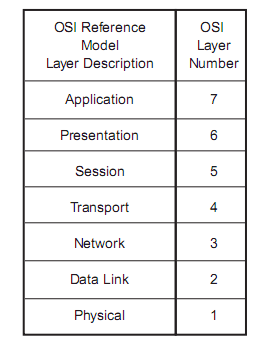


Figura - Modelo de referência OSI

Fonte: Siyan et al (2002, p. 11)

Ainda Siyan et al (2002) diz que essas camadas possuem as seguintes funções:

* Physical: é a camada de mais baixo nível e define a forma como os *bits* serão transferidos entre os computadores. É nesse nível que as tensões (volts) que representam os *bits*, a forma de controlar o acesso ao meio de comunicação, a frequência de oscilação dos sinais e outras operações de baixo nível são definidas;
* Data Link: é responsável por empacotar os dados em um nível mais baixo e garantir a validade dos dados. Ainda nessa camada endereços físicos são definidos, por exemplo, o padrão Ethernet define um endereço que é fixo para cada placa de rede. Esse endereço se encontra nessa camada e é fundamental para a comunicação em redes Ethernet;
* Network: essa camada já não é mais implementada via hardware, mas sim pelo sistema operacional. Nessa camada são definidos endereços lógicos utilizados no processo de comunicação. O protocolo IP se encontra nessa camada e é responsável por transferir os pacotes de dados entre a origem e destino;
* Transport: efetua o encapsulamento dos dados, a decomposição de um grande volume de dados em vários pacotes diferentes e garante que os mesmos são recebidos na ordem no destino. É nessa camada que o protocolo TCP é definido e também as portas de comunicação utilizadas para representar cada serviço;
* Session: é uma camada não muito utilizada, mas que tem por objetivo controlar o início e o fim de sessão de comunicação. Muitos protocolos controlam essas sessões diretamente na camada de transporte;
* Presentation: é responsável por tratar a apresentação dos dados entre os diferentes tipos de computadores existentes. Por exemplo, *mainframes* utilizam a ordem dos bytes (em estruturas com mútiplos bytes) de forma invertida que os computadores desktop, portanto é necessário um processo de conversão para que a comunicação ocorra com sucesso;
* Application: é a camada final e representa a aplicação efetuando a transmissão ou recebendo os dados. Essa aplicação utiliza um protocolo de comunicação para saber a ordem do envio das mensagens, parâmetros necessários ou opcionais, entre outros. Nessa camada se encontra o popular protocolo HTTP utilizando em páginas web. Além desse protocolo temos o FTP, SSL, SIP e muitos outros.

Conforme mostrado o modelo OSI aborda todos os aspectos da comunicação entre dois computadores. Além disso, nesse modelo em camadas, é possível substituir uma camada por outra de forma de forma transparente para as camadas superiores, desde que a camada substituída utilize a mesma interface de comunicação.

## Banco de Dados

Hoffer (2007, p. 6) define um banco de dados como “*a database is an organized collection of logically related data*”. Com base nessa definição é possível verificar que os dados devem estar armazenados de forma organizada e também relacionados.

Esses dados, armazenados no banco de dados, são organizados em tabelas que são compostas de atributos e os relacionamentos entre essas entidades são definidos utilizando atributos comuns entre elas.

### Entidades

Hoffer (2007, p. 96) ainda define uma entidade, também conhecida como tabela, como “*an entity is a person, place, object, event or concept in the user environment about which the organization wishes to maintain data*”. Ou seja, as entidades representam os **objetos** que interagem no domínio do problema sendo sistematizado.

Alguns exemplos de entidades:

1. Empresa: representa as empresas com as quais o sistema interage;
2. Colaborador: representa os funcionários da organização;
3. Computador: no contexto do netHPC representa um computador onde o processamento pode ser distribuído.

### Atributos

Uma entidade é composta por atributos, que segundo Hoffer (2007, p 100) é “*an attribute is a property or characteristic of an entity type that is of interest to the organization*”.

Seguindo com os exemplos do tópico de entidades os atributos para entidades seriam:

1. Empresa: CNPJ, nome, endereço, segmento de mercado;
2. Colaborador: CPF, nome, tipo sanguíneo, cargo, salário;
3. Computador: nome, endereço na rede, configuração de hardware;

### Modelo de Entidades e Relacionamentos

O modelo de entidades e relacionamentos (MER) é utilizado para modelar as entidades e os relacionamentos entre estas. Como qualquer modelo, o MER torna mais fácil a representação e validação das entidades e, como Hoffer (2007, p 93) diz “*an entity-relationship model is a detailed, logical representation of the data for an organization or for a business area*”, ou seja, o MER apresenta uma visão lógica e detalhada das entidades e relacionamentos destas de um determinado sistema ou área da organização. Na é possível visualizar um exemplo de modelo de entidades e relacionamentos na notação gráfica utilizada na empresa onde o estágio foi efetuado.

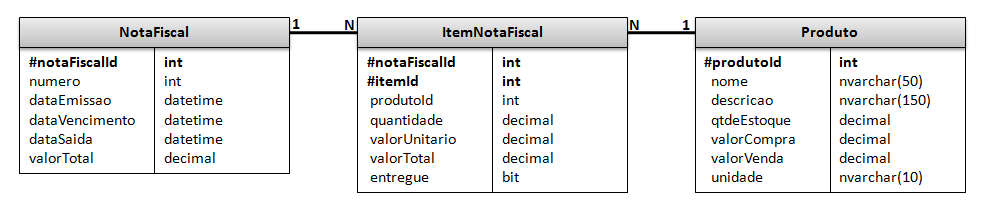


Figura - Modelo de Entidades e Relacionamentos

Fonte: George Luiz Bittencourt (2009)

### Normalização de Dados

Ao elaborar a estrutura necessária para armazenar os dados das entidades definidas o problema da redundância de dados surge. Para resolver esse problema a técnica da normalização de dados é aplicada. Essa técnica tem por objetivo remover a duplicidade dos dados dividindo esses dados em novas entidades e relacionando estas.

Segundo Powell (2006) para se atingir a normalização de dados, e com isso reduzir a redundância de dados, três formas normais devem ser aplicadas em sequência. Essas formas normais podem ser visualizadas na sequência.

* Primeira Forma Normal (1NF): uma entidade é dita estar na primeira forma normal quando em seus atributos não existem listas ou conjuntos de dados. Um exemplo dessa situação é uma entidade **Cliente** onde nessa mesma entidade existe uma lista de telefones. Para resolver esse problema uma nova entidade deve ser criada e esses telefones devem ser movidos para ela e um relacionamento deve ser estabelecido;
* Segunda Forma Normal (2NF): é quando a entidade se encontra na primeira forma normal e todo atributo não chave da entidade depende somente da chave primária da mesma;
* Terceira Forma Normal (3NF): quando a entidade se encontra na segunda forma normal e não existe uma dependência entre atributos não chave primária da entidade.

### Sistema Gerenciador de Bancos de Dados

Segundo Hoffer (2007, p. 8), um sistema gerenciador de banco de dados (SGBD ou RDBMS em inglês) é “*a software system that is used to create, maintain and provide controlled access to user databases*”.

Ainda segundo Hoffer (2007), as principais tarefas de um SGBD são:

* Gerenciar a alocação dos dados no meio de armazenamento utilizado. Um exemplo dessa tarefa é o de alocar espaço em um arquivo quando um novo registro é inserido na tabela de clientes;
* Garantir que as transações concorrentes não corrompam os dados e sejam persistidas mesmo quando um problema de hardware ocorrer. Por exemplo, se durante a inserção de um registro ocorrer uma queda de energia, o SGBD precisa garantir que no retorno os dados estarão íntegros;
* Controlar o acesso dos usuários com base nas permissões definidas. Um usuário não pode acessar uma tabela para o qual não tenha permissão, por exemplo: um usuário do departamento de informática não pode consultar a tabela de funcionários devido às informações contidas nestas;
* Executar as consultas com o melhor desempenho possível utilizando os recursos do hardware, como memória, disco e processador da melhor forma possível. Ainda nesse sentido, utilizando os índices criados pelo desenvolvedor do sistema.

No mercado atual existem diversos SGBDs disponíveis, como por exemplo: SQL Server, Oracle, DB2, mySQL e muitos outros. Alguns destes são de distribuição gratuita, porém a grande maioria necessita de licenciamento.

## Engenharia de Software

Segundo Sommerville (2006, p. 6), engenharia de software é “*an engineering discipline which is concerned with all aspects of software production*”. Ou seja, a engenharia de software é a divisão da engenharia que trata de todos os aspectos da construção de um software. Afinal, construir um software não é somente codificar o mesmo, mas também gerar a documentação de utilização, especificação de construção, efetuar a manutenção do mesmo quando em campo, e outras atividades.

Construir algo significa seguir uma série de passos até obter o resultado final. A esses passos é dado o nome de processo, que segundo Pressman (2006, p. 16) é “um roteiro que o ajuda a criar a tempo um resultado de alta qualidade”. Esse roteiro, citado por Pressman, é chamado de processo de desenvolvimento de software e seu objetivo é definir os passos pelos quais o desenvolvimento de um software deve passar para ser produzido com qualidade e dentro do prazo e orçamento previsto.

Em relação aos passos que compreendem um processo de desenvolvimento de software, Bezerra (2006, p. 20) afirma:

Cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de um desenvolvimento. Entretanto, podem-se distinguir atividades que, com uma ou outra modificação, são comuns à maioria dos processos existentes.

Ainda segundo Bezerra (2006) esses passos são:

* Levantamento de Requisitos: é a inicial do desenvolvimento de um software e visa entender o problema que o software deve atender. O atendimento desse problema é efetuado através da definição de requisitos. Esses requisitos por sua vez são definidos em conjunto com o solicitante do software. Um requisito é algo que o software deve atender, como por exemplo, manutenir clientes é um requisito que um software de controle de vendas precisa atender a fim de resolver o problema de controle de vendas;
* Análise de Requisitos: com o resultado da fase de levantamento de requisitos, que é o que o sistema deve atender, os analistas criam modelos em uma especificação a fim de definir como esses requisitos devem ser atendidos. Da mesma forma que no levantamento de requisitos não havia a preocupação formal com a tecnologia, na fase de análise também ainda não se leva ela em consideração.
* Projeto: nessa fase a pergunta “como o sistema será implementado” é respondida. Preocupações como, que linguagem de programação será utilizada, qual sistema gerenciador de banco de dados utilizar ou ainda se o sistema terá sua apresentação em forma web são definidas nessa fase. Essa fase se preocupa em definir quais recursos tecnológicos disponíveis serão utilizados para atender os requisitos impostos pela fase de levantamento de requisitos;
* Implementação: é aqui que a codificação do sistema é efetuada. O programador, com base nas especificações geradas pelas fases anteriores, transforma essas em um sistema;
* Testes: é a fase onde o resultado da fase de implementação é comparado com o que foi especificado, ou seja, são efetuados testes a fim de validar se o código construído está de acordo com as especificações;
* Implantação: é a fase final do processo de desenvolvimento de software e tem por objetivo implantar o sistema criado na fase de implementação no ambiente do cliente para utilização. Nessa fase são ministrados treinamentos aos usuários, conversões de bases de dados, e outras atividades necessárias para a implantação da solução.

Ao analisar essas fases deve-se perceber que conforme o processo vai caminhando, o nível de abstração vai diminuindo. Inicialmente, na fase de levantamento de requisitos, tinha-se uma definição bem preliminar do que precisava ser desenvolvido. Conforme as fases de análise, projeto e implementação foram sendo executadas, o nível de abstração foi diminuindo, chegando ao ponto de na fase de implementação o programador estar escrevendo o código para atender ao requisito.

Durante o desenvolvimento do software, artefatos são gerados a fim de documentar o que foi requisitado e também com o objetivo de repassar a necessidade do cliente para as demais partes que compõem o desenvolvimento.

Esses artefatos gerados são os documentos com os requisitos, modelos, diagramas, códigos fontes e outros. Na modelagem do sistema são utilizados diagramas de alguma ferramenta. Dependendo do paradigma de software sendo utilizado no desenvolvimento, uma ou outra ferramenta é utilizada. No paradigma orientado a objetos a linguagem UML é utilizada, pois com seus diferentes diagramas é possível definir os objetos e interações entre estes.

## UML – *Unified Modeling Language*

Conforme Eriksson (2004, p. 1), a UML provê “*industry standard mechanisms for visualizing, specifying, constructing and documeting software systems*”. Com base nisso e ao analisar o significado da sigla UML em português, linguagem de modelagem unificada, podemos entender o seu objetivo. Ela é uma linguagem, pois através das suas notações gráficas podemos expressar idéias, é focada em criar modelos de software e é unificada, pois tenta ser a linguagem comum a todos, evitando que cada um utilize suas notações dificultando o processo de comunicação.

Segundo Rumbaugh (2006, p. 18), a UML se caracteriza por representar um software a partir de três pontos de vista diferentes, conhecidos como modelos. Para cada modelo existe um ou mais diagramas nos quais elementos gráficos são dispostos para representar um aspecto do sistema pretendido. Na sequência, é possível verificar esses modelos e os diagramas relacionados a eles.

* Modelo de Classes: modela a estrutura dos objetos envolvidos no domínio do problema e suas relações. Serve de base para os demais modelos e é constituído do diagrama de classes;
* Modelo de Estados: mostra os estados pelos quais um objeto pode transitar e as condições para essas transições. Esse modelo é representado pelo diagrama de estados;
* Modelo de Interações: documenta as interações entre os objetos que compõem o sistema. Os diagramas desse modelo são o de casos de uso, de sequência e de atividades.

É importante salientar, que da mesma forma que uma casa possui um diagrama elétrico, um diagrama hidráulico e outros, um software também possui essa necessidade, afinal, um software é uma combinação de vários itens diferentes que interagem entre si para proporcionar as funcionalidades que atendem os requisitos impostos pelos usuários.

Na é possível visualizar um diagrama de casos de uso de exemplo. É possível verificar os atores envolvidos que iniciam ou interagem com os casos de uso. É possível visualizar também o sistema, representado graficamente por um retângulo e os casos de uso oferecidos pelo sistema aos atores.

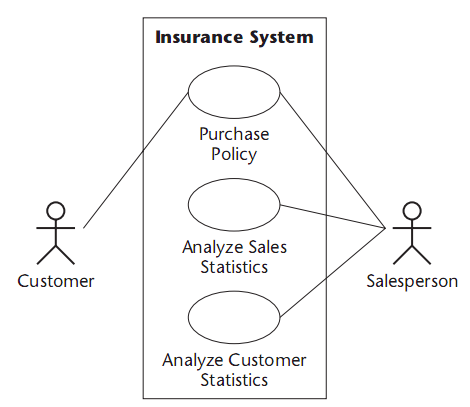


Figura - Diagrama de casos de uso

Fonte: Eriksson 2004, p. 25

Já na há a representação dos objetos de um sistema através do diagrama de classes. Esse diagrama mostra as classes, que modelam os objetos do sistema e as relações entre esses objetos.

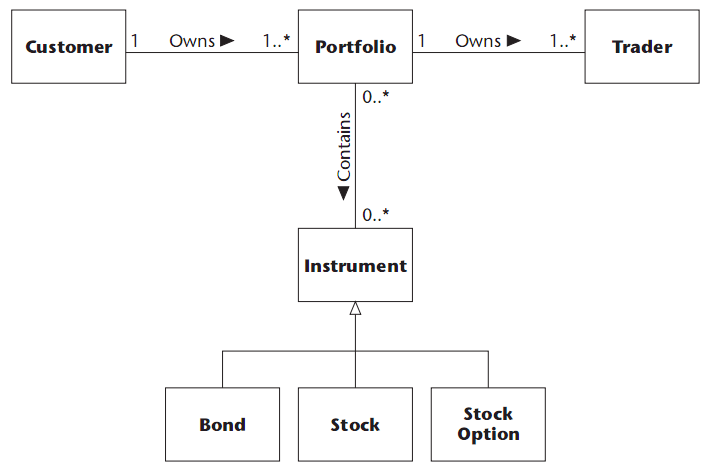


Figura - Diagrama de classes

Fonte: Eriksson 2004, p. 26

Finalmente, a mostra os estados pelo qual um elevador pode transitar durante o seu funcionamento. Esse diagrama é utilizado para modelas os estados de um sistema.

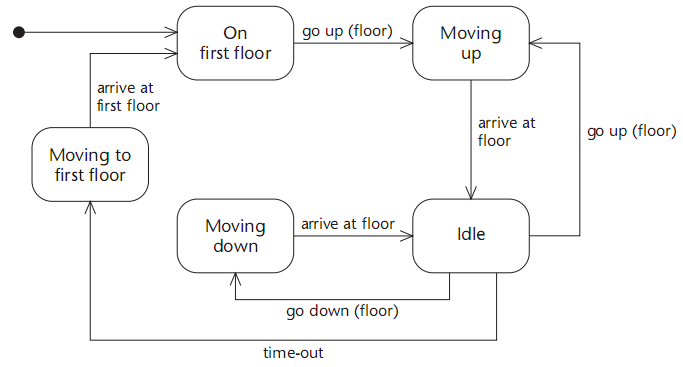


Figura - Diagrama de estados

Fonte: Eriksson 2004, p. 25

Os fundamentos apresentados nesse capítulo serviram de base para o desenvolvimento da prática do estágio, auxiliando o acadêmico na construção do *middleware* de computação em grade.

# DESCRIÇÃO DA PRÁTICA

Neste capítulo final, o detalhamento do desenvolvimento da solução, bem como os artefatos gerados, é apresentado. Os diagramas de classe e caso de uso, o modelo de entidades e relacionamentos, as telas construídas e o resultado dos testes da solução em campo são mostrados e explicados.

O nome **netHPC**, nome dado ao *middleware* pelo acadêmico, surgiu da aglutinação das palavras **net** e **HPC**, onde **net** é um prefixo ou sufixo comumente utilizado em produtos que utilizam a plataforma Microsoft .NET, e **HPC** vem do inglês *high performance computing*, que é o termo que designa a computação de alto desempenho.

Como um dos objetivos do acadêmico é disponibilizar o netHPC em forma de código fonte aberto, foi escolhido o idioma inglês para a terminologia utilizada. Isso foi feito em todas as classes, tabelas, telas e outros componentes. O idioma inglês foi escolhido, pois é considerado o idioma universal.

## Local do Desenvolvimento

O estágio curricular foi realizado na Integral Informática Ltda, empresa no qual o acadêmico já atua como consultor tecnológico à 8 anos. A Integral Informática atua a mais de 13 anos no segmento de consultoria em soluções Microsoft, Citrix e Linux na região norte catarinense. Foi a primeira parceira Microsoft em Santa Catarina e hoje o seu nível de parceria com esta é *gold*. Além dessa parceria a Integral é ainda parceira s*ilver* da Citrix no ramo de entrega e virtualização de aplicativos. Com essas parcerias e os mais de 15 profissionais certificados nas mais diversas áreas, a competência da empresa perante o mercado é atestada.

A Integral divide-se em quatro áreas de negócio, sendo: **Integração**, **Soluções de Negócio**, **Licenciamento** e **Outsourcing**. Essas áreas oferecem os seguintes serviços ao mercado:

* **Integração**: integração de tecnologias Microsoft, Citrix e Linux formando cenários para as mais diversas solicitações;
* **Soluções de Negócio**: desenvolvimentos customizados e implantação de soluções de ERP e CRM da Microsoft;
* **Licenciamento**: comercialização de licenças dos produtos Microsoft e Citrix;
* **Outsourcing**:terceirização da administração dos ambientes computacionais e contratos de horas de suporte.

Com a sinergia de todas essas áreas, a Integral consegue prover “**Soluções de Integração, Outsourcing, Licenciamento de Software e Negócio que agreguem Valor e proporcionem Tranquilidade para os nossos Clientes**”, que é o negócio da empresa.

A Integral orgulha-se de atender clientes de renome nacional e internacional, como a Bunge Alimentos, a Amanco Brasil, a Comercial Salfer, as Duas Rodas Industrial e muitas outras. Além disso, a Integral já executou projetos fora do Brasil, como por exemplo, na Suíça, em Cingapura, nos Estados Unidos, na Argentina e outros.

O coordenador de campo foi o Sr. Carlos Eduardo Bittencourt Fersura, que ocupa o cargo de presidente da empresa e lidera a área de Integração, área na qual foi efetuado o estágio.AIntegral oportunizou ao acadêmico a experiência de executar um trabalho de pesquisa de ponta, pois isso vai de encontro a um dos valores que a empresa mais acredita, que é o da **inovação constante**.

## Desenvolvimento da Solução

### Requisitos Funcionais

Ao analisar as soluções de computação em grade existentes, os requisitos funcionais da sequência foram definidos para serem atendidos durante o estágio.

No é possível visualizar o requisito funcional que determina a necessidade de uma interface que o algoritmo deve implementar para utilizar os serviços oferecidos pelo *middleware*.

|  |  |
| --- | --- |
| **Identificador** | REQFUNC-01 |
| **Nome** | Prover uma interface que deve ser utilizada pelos algoritmos |
| **Definição** | A solução deve definir uma interface que os algoritmos que serão distribuídos devem implementar para serem utilizados no netHPC. |

Quadro - Requisito funcional REQFUNC-01

Fonte: George Luiz Bittencourt (2009)

O mostra o requisito funcional no qual fica estabelecido que a administração deverá ser centralizada.

|  |  |
| --- | --- |
| **Identificador** | REQFUNC-02 |
| **Nome** | Execuções devem ser iniciadas e abortadas a partir de um computador central |
| **Definição** | Os especialistas devem utilizar um computador central, sem ter a necessidade de se deslocar até os computadores utilizados durante o processamento. |

Quadro - Requisito funcional REQFUNC-02

Fonte: George Luiz Bittencourt (2009)

Já no é possível visualizar o requisito que define que é necessário determinar os pontos que mais influenciam no tempo total de processamento com o objetivo de fornecer dados para otimizar o algoritmo ou ainda o netHPC.

|  |  |
| --- | --- |
| **Identificador** | REQFUNC-03 |
| **Nome** | O *middleware* deve prover uma forma de mensurar o tempo despendido em seu código e no código do algoritmo |
| **Definição** | O *middleware*, da mesma forma que o algoritmo, possui instruções que são executadas e essas por sua vez influenciam no tempo total de processamento. Com base nisso é importante mensurar o tempo despendido tanto em código do *middleware* quanto em código do algoritmo. |

Quadro - Requisito funcional REQFUNC-03

Fonte: George Luiz Bittencourt (2009)

No fica estabelecida a necessidade do agente efetuar a sua atualização de forma automática.

|  |  |
| --- | --- |
| **Identificador** | REQFUNC-04 |
| **Nome** | O software instalado nos computadores deve se atualizar automaticamente |
| **Definição** | O agente que será instalado em cada computador deve-se atualizar automaticamente eliminando a necessidade do especialista ter que se deslocar até o computador para aplicar atualizações de software. |

Quadro - Requisito funcional REQFUNC-04

Fonte: George Luiz Bittencourt (2009)

O estabelece que o especialista não deve ter que criar uma entrada para os computadores previamente, mas sim que o agente deve efetuar essa operação automaticamente.

|  |  |
| --- | --- |
| **Identificador** | REQFUNC-05 |
| **Nome** | Os computadores devem entrar automaticamente na grade de computadores |
| **Definição** | Uma vez o agente iniciado em um computador o mesmo deve automaticamente criar o seu registro, sem a necessidade de um especialista ter que efetuar essa operação. |

Quadro - Requisito funcional REQFUNC-05

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar o requisito final onde fica estabelecida a necessidade de um relatório para verificar o resultado das execuções.

|  |  |
| --- | --- |
| **Identificador** | REQFUNC-06 |
| **Nome** | Prover um relatório onde o especialista possa visualizar o resultado das execuções |
| **Definição** | O especialista, a partir de um computador central, deve poder visualizar o resultado das execuções através de um relatório. Esse relatório deve mostrar o tempo para processar cada item de trabalho e também o impacto do *middleware* no tempo total de processamento. |

Quadro - Requisito funcional REQFUNC-06

Fonte: George Luiz Bittencourt (2009)

#### Modelo de Domínio

Ao analisar o domínio do problema que se deseja resolver, a construção de um *middleware* de computação em grade, e os requisitos funcionais encontrados, as entidades listadas no diagrama de domínio da foram definidas para suportar a distribuição do algoritmo. Todas essas classes se encontram no pacote **netHPC.Shared** e são utilizadas em vários pontos da solução.

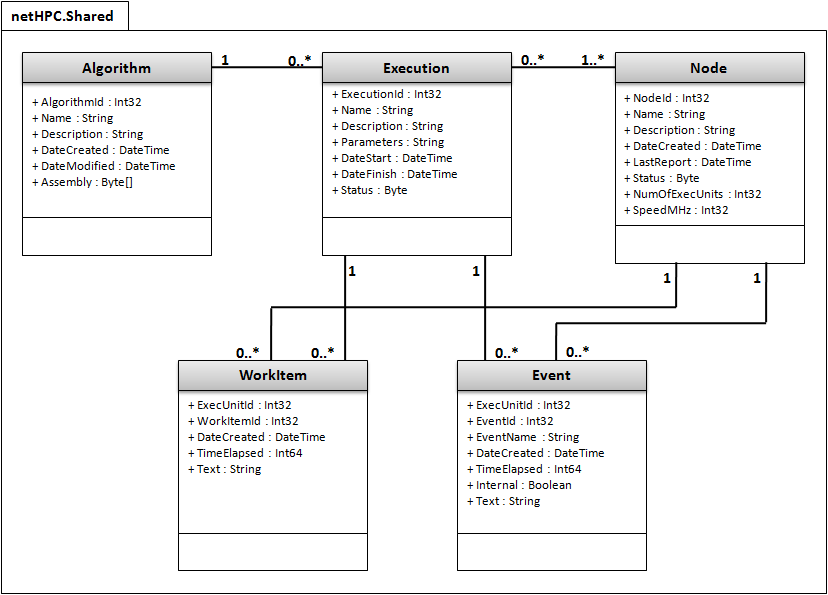


Figura - Diagrama de domínio do netHPC

Fonte: George Luiz Bittencourt (2009)

A entidade **Algorithm** representa um algoritmo disponível para distribuição através da solução. Além dos atributos padrões como nome, descrição e datas de criação e modificação, a entidade tem também o atributo **Assembly**. Através desse atributo é possível acessar o código compilado do algoritmo desenvolvido em .NET. Esse mesmo código é distribuído e executado nos *computing nodes* quando uma execução é iniciada.

A entidade **Node** representa os computadores que podem ser utilizados na distribuição da execução do algoritmo. As ocorrências dessa entidade são criadas automaticamente pelo agente instalado em cada computador. O atributo **LastReport** indica a data e hora do último reporte do computador ao *middleware*. O atributo **Status** indica o estado do agente, podendo assumir os valores **Idle**, **Executing** ou **Unknow**. Os atributos **NumOfExecUnits** e **SpeedMHz** indicam respectivamente o número de *cores* disponíveis para utilização e a velocidade destes.

Tendo o algoritmo definido e no mínimo um *computing node*, já é possível iniciar uma execução através do *middleware*. Essa execução é representada pela entidade **Execution**. Essa entidade se relaciona com a entidade **Algorithm**, indicando a qual algoritmo a execução se relaciona e também se relaciona com os **Nodes** utilizados durante a execução. Essa entidade armazena os parâmetros da execução no atributo **Parameters** e o status da execução no atributo **Status**, sendo que este último pode assumir os valores **Executing**, **Finished** ou ainda **Cancelled**.

O resultado do processamento de um *work item* é representado pela classe **WorkItem**. Essa classe possui o atributo **TimeElapsed** que armazena o tempo total para processar o *work item* em milisegundos e o atributo **Text** que armazena o resultado da execução. O conteúdo desse atributo e a interpretação do mesmo são de responsabilidade do algoritmo executado, o *middleware* simplesmente armazena o valor sem efetuar qualquer análise do mesmo.

A entidade final é a **Event** que é utilizada para mensurar eventos que ocorrem durante o processamento. Esses eventos podem ser internos, ou seja, gerados pelo *middleware* ou ainda eventos do algoritmo. Para determinar se um evento é interno ou do algoritmo, o atributo **Internal** é utilizado. Através do atributo **EventName** é possível determinar em que parte do código o mesmo foi gerado. É através dos eventos internos que o netHPC calcula o *overhead* do mesmo sobre o tempo total de execução.

Para que o algoritmo possa ser distribuído pelo *middleware* ele deve estar de acordo com uma especificação definida por este. A mostra o diagrama de classes do kit de desenvolvimento (SDK, do inglês software development kit) que o algoritmo deve implementar para que o mesmo possa ser distribuído. Todas essas classes e interfaces estão definidas no pacote **netHPC.SDK**.

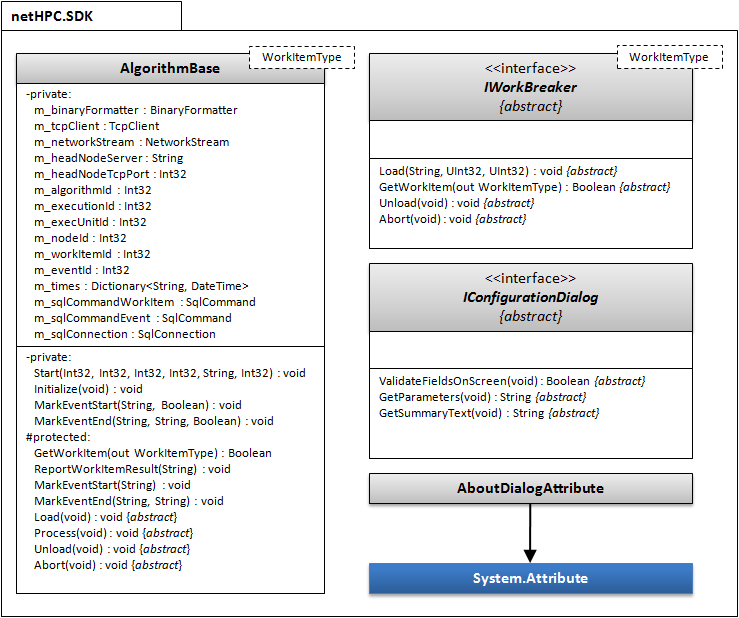


Figura - Diagrama de classes do kit de desenvolvimento do netHPC

Fonte: George Luiz Bittencourt (2009)

A interface **IConfigurationDialog** é implementada por uma classe no algoritmo para apresentar uma interface gráfica onde o usuário pode parametrizar a execução do mesmo. A classe deve implementar todos os métodos para que possa ser utilizada e o objetivo de cada método está detalhado na sequência.

* **ValidateFieldsOnScreen**: o *middleware* executa esse método antes de iniciar a execução para que o algoritmo possa validar se os parâmetros informados na tela pelo usuário estão corretos;
* **GetParameters**: o *middleware* executa esse método para receber os parâmetros selecionados pelo usuário no formato definido pelo algoritmo. Esses parâmetros são utilizados posteriormente pela classe que implementa a interface **IWorkBreaker**. O retorno desse método é armazenado no parâmetro **Parameters** da entidade **Execution**;
* **GetSummaryText**: o *middleware* executa esse método para receber um texto resumindo os parâmetros que é mostrado em tela antes de iniciar a execução.

A interface **IWorkBreaker** define os métodos que a classe responsável por quebrar o trabalho total deve implementar. Essa classe é executada no *head node* e o objetivo de cada método é detalhado na sequência.

* **Load**:é o primeiro método a ser executado e tem por objetivo inicializar a classe. Esse método recebe como argumento os parâmetros retornado pelo método **GetParameters** da interface **IConfigurationDialog** e o número de computadores e *cores* selecionados para a execução;
* **GetWorkItem**: é executado a cada requisição de um *work item* pelos *computing nodes* e deve retornar uma instância de uma classe que representa uma fatia do trabalho total;
* **Unload**: é o último método a ser executado e tem por objetivo liberar possíveis recursos alocados pela classe;
* **Abort**: é executado quando o usuário aborta uma execução através da console de gerenciamento e tem por objetivo sinalizar para a classe essa situação para que a mesma execute algum código específico necessário nesse caso.

Algoritmos podem ser construídos por diversas entidades diferentes como universidades, empresas, professores, etc. Estes podem ser comerciais ou de distribuição gratuita. Pensando nisso e na proteção da propriedade intelectual de cada um, o *middleware* possibilita o algoritmo definir uma tela indicando os direitos autorais do mesmo. Para que isso seja possível o algoritmo deve implementar uma classe de controle gráfico e marcar essa classe com o atributo **AboutDialogAttribute**. Com isso sempre que o usuário acessar as propriedades do algoritmo na console de gerenciamento o conteúdo dessa classe será mostrada ao usuário.

A última classe é a mais importante e se chama **AlgorithmBase**. Essa classe deve ser herdada pelo algoritmo a fim de utilizar os métodos providos por essa e sua execução ocorre em cada *computing node*. Os métodos que essa classe oferece ao algoritmo estão detalhados na sequência.

* **Start**:éum método interno do netHPC e é utilizado para iniciar a execução em um core;
* **Initialize**: é um outro método interno utilizado para se conectar no *data store* e no *head node*;
* **GetWorkItem**: é o método utilizado pelo algoritmo para requisitar um work item para ser processado. Quando esse método é invocado uma chamada pela rede é feita ao *head node*. No retorno desse método um *work item* é recebido ou uma indicação é dada de que não existem mais *work items* para processamento.
* **ReportWorkItemResult**: é utilizado pelo algoritmo para reportar o resultado do processamento do *work item*. Esse resultado é persistido no *data store* na entidade **WorkItem**;
* **MarkEventStart**: é utilizado para marcar o início de um evento que se deseja mensurar. Ao ser invocado a data e hora do evento é armazenada em memória para uso posterior;
* **MarkEventEnd**: marca o fim de um evento previamente iniciado. Quando invocado a diferença de tempo entre o início e o fim é calculada e armazenada no *data store* na entidade **Event**;
* **Load**: é o primeiro método a ser executado no início de uma execução e seu objetivo é inicializar variáveis internas ou ainda executar outras tarefas a fim de suportar o processamento do algoritmo. Esse método pode ser sobrescrito pela subclasse, pois sua utilização é opcional;
* **Process**: esse método é implementado pelo algoritmo e é executado pelo *middleware* para iniciar o processamento do algoritmo em si. É nesse método que a lógica que irá processar os *work items* deve estar localizada. É executado logo após o método **Load** e esse método deve ser sobrescrito pela subclasse. Sua utilização é obrigatória;
* **Unload**: é executado após o término do processamento do algoritmo e seu objetivo é desfazer possíveis alocações efetuadas pelo método **Process** ou **Load**. Esse método pode ser sobrescrito pela subclasse, seu uso é opcional;
* **Abort**: é executado somente quando o usuário aborta a execução a partir da console de gerenciamento. Seu objetivo é abortar a execução do algoritmo e deve ser sobrescrito pela subclasse somente quando for necessário ser notificado desse acontecimento.

A classe **AlgorithmBase** e a interface **IWorkBreaker** são parametrizadas utilizando o recurso *generics* existente nas linguagens de programação orientadas a objeto. Esse recurso permite parametrizar essa classe e interface com um tipo de dado em tempo de compilação, dessa forma a classe e interface geradas utilizam esse tipo de dado diretamente no código. No diagrama o **WorkItemType** é o nome utilizado para parametrizar esse tipo de dado, que será utilizado como um *work item*. Com esse recurso é possível utilizar tipos de dados fortemente tipados, evitando com isso erros de programação, uma vez que a utilização desses tipos de dados são validadas em tempo de compilação e também aumentando o desempenho do aplicativo, pois não são efetuadas conversões em tempo de execução.

#### Regras de Negócio

Ao analisar a estrutura definida no modelo de domínio, os requisitos funcionais e principalmente o domínio do problema, as regras de negócio na sequência foram definidas.

O define como cada *core* deve ser contabilizado pelo *middleware*.

|  |  |
| --- | --- |
| **Identificador** | RN-01 |
| **Nome** | Cada *core* corresponde a uma unidade de processamento |
| **Definição** | Computadores que possuírem mais de um processador ou ainda um único processador com mais de core devem retornar o número total de *cores*. Exemplo:  Dual Quad-Core: 2 \* 4 = 8 cores disponível para o netHPC |

Quadro - Regra de negócio RN-01

Fonte: George Luiz Bittencourt (2009)

Já o define que o peso do *middleware* deve ser o menor possível.

|  |  |
| --- | --- |
| **Identificador** | RN-02 |
| **Nome** | O *overhead* deve ser o mais baixo possível |
| **Definição** | O peso adicional que o netHPC adiciona ao processamento do algoritmo deve ser o menor possível a fim de justificar sua utilização. |

Quadro - Regra de negócio RN-02

Fonte: George Luiz Bittencourt (2009)

#### Glossário do Sistema

Os termos na sequência se aplicam a computação de alto desempenho e ao netHPC, e são utilizados extensivamente nesse trabalho.

* **agente**: software executado tanto no *head node* quanto no *computing node* que recebe e processa as requisições de processamento vindas pela rede.
* **algoritmo**:lógica que se deseja distribuir para os *computing nodes* e tem por objetivo simular um fenômeno;
* **computing node**: todo computador utilizado para o processamento do algoritmo;
* **console de gerenciamento**:software executado em qualquer computador e que permite cadastrar algoritmos, iniciar execuções e visualizar relatórios de execução;
* **data store**: banco de dados relacional onde os algoritmos, os resultados das execuções e os eventos são armazenados;
* **evento**: artifício utilizado para mensurar o tempo despendido em eventos internos do netHPC ou ainda em eventos definidos pelo algoritmo sendo executado;
* **interface**: é uma forma de ligação entre duas entidades. No contexto do netHPC, um exemplo seria o kit de desenvolvimento de software que é a interface que os algoritmos utilizam para usufruir das funcionalidades do mesmo;
* **head node**: computador que distribui os *work itens* para processamento entre os *computing nodes*;
* **kit de desenvolvimento**: conjunto de classes e interfaces que o algoritmo deve implementar para que sua execução possa ser distribuída entres os *computing nodes*.
* **middleware**: é o software que gerencia a distribuição e execução dos algoritmos e também fornece um conjunto de APIs que o algoritmo deve utilizar para se beneficiar do mesmo;
* **overhead**: toda carga adicional não relacionada diretamente ao processamento do algoritmo, mas que influencia no tempo de execução do mesmo. Alguns exemplos de *overhead* são os tempos despendidos em comunicações de rede, tempos no *middleware*, tempos em chamadas ao sistema operacional, entre outros;
* **thread**: unidade básica de execução do sistema operacional Windows. Todo algoritmo quando executado inicia com uma única *thread* e essa por sua vez pode iniciar outras. É no contexto de uma thread que as instruções de um programa são executadas;
* **work item**: resultado do processo de decomposição do trabalho total e distribuído para processamento nos *computing nodes*. O resultado desse processamento é armazenado no *data store* e pode ser visualizado pelo especialista.

#### Atores

Após analisar a estrutura do *middleware* que se deseja implementar os atores foram identificados e suas informações foram detalhadas na sequência.

O ator do é o especialista com o conhecimento sobre o problema que se deseja resolver ou sobre fenômeno que se deseja estudar. Esse ator irá interagir com o sistema analisando o resultado das execuções, criando algoritmos com o kit de desenvolvimento para serem posteriormente cadastrados no netHPC e também iniciando e parando execuções do algoritmo.

|  |  |
| --- | --- |
| **Identificador** | ATOR-01 |
| **Nome** | Especialista |
| **Definição** | São os usuários com o conhecimento específico sobre o problema ou fenômeno que um algoritmo modela. |
| **Permissão de Acesso** | Controle total |
| **Freqüência de Uso** | Diária |
| **Nível de Instrução** | Alta |
| **Proficiência na**  **Aplicação** | Alta |
| **Proficiência em**  **Informática** | Alta |

Quadro - Ator ATOR-01

Fonte: George Luiz Bittencourt (2009)

Já o ator do representa o sistema operacional onde o agente é executado. Esse ator irá iniciar e parar o agente.

|  |  |
| --- | --- |
| **Identificador** | ATOR-02 |
| **Nome** | Windows |
| **Definição** | Representa o sistema operacional Windows sobre o qual o agente é executado. |
| **Permissão de Acesso** | Iniciar e parar o agente |
| **Freqüência de Uso** | Diária |
| **Nível de Instrução** | Não se aplica |
| **Proficiência na**  **Aplicação** | Não se aplica |
| **Proficiência em**  **Informática** | Não se aplica |

Quadro - Ator ATOR-02

Fonte: George Luiz Bittencourt (2009)

Finalmente o ator do representa um *timer* interno do netHPC. Esse ator irá disparar periodicamente eventos para verificar se existem atualizações disponíveis.

|  |  |
| --- | --- |
| **Identificador** | ATOR-03 |
| **Nome** | Relógio |
| **Definição** | Ator utilizado para iniciar casos de uso periódicos. |
| **Permissão de Acesso** | Atualizar o agente |
| **Freqüência de Uso** | A cada 30 minutos |
| **Nível de Instrução** | Não se aplica |
| **Proficiência na**  **Aplicação** | Não se aplica |
| **Proficiência em**  **Informática** | Não se aplica |

Quadro - Ator ATOR-03

Fonte: George Luiz Bittencourt (2009)

#### Modelo de Casos de Uso

Soluções de *grid computing* complexas e que se encontram a muitos anos no mercado são compostas de diversos casos de uso, entretanto devido o caráter experimental do estágio, o netHPC irá iniciar com os casos de uso apresentados no diagrama da .

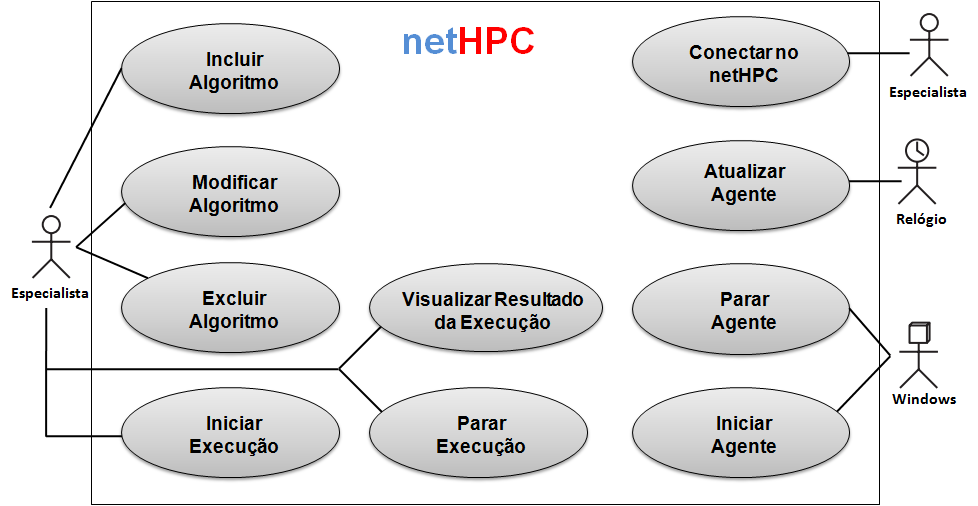


Figura - Diagrama de casos de uso

Fonte: George Luiz Bittencourt (2009)

Uma solução de computação em grade segue basicamente um ciclo onde um algoritmo para modelar determinado fenômeno é construído, em seguida o mesmo é distribuído para processamento e o resultado desse processamento é analisado. Com base nesse resultado o algoritmo pode ser melhorado para diminuir o tempo ou ainda eventuais erros programação podem ser corrigidos.

Ao analisar esse ciclo de utilização os casos de uso foram definidos. Os casos de uso **Incluir Algoritmo, Modificar Algoritmo e Excluir Algoritmo** são utilizados para efetuar o cadastro dos algoritmos que serão distribuídos no *data store*. Os casos de uso **Iniciar Execução** e **Parar Execução** são utilizados para gerenciar a execução entre os computadores. E finalmente o caso de uso **Visualizar Resultado da Execução** é onde o especialista verifica o resultado do algoritmo e decide se o mesmo dever ser corrigido ou melhorado. Os demais casos de uso definidos foram criados somente para suportar esse ciclo e os requisitos funcionais.

#### Priorização dos Casos de Uso

Todos os casos de uso presentes no diagrama da foram construídos durante o período de estágio.

#### Detalhamento dos Casos de Uso

O detalhamento dos casos de uso mostrados anteriormente na podem ser visualizados na sequência.

No os detalhes do caso de uso utilizado para conectar no netHPC pode ser visualizado.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-01 |
| **Nome** | | Conectar no netHPC |
| **Objetivo** | | Conectar no *data store* e no *head node* do netHPC possibilitando com isso iniciar, parar e manutenir algoritmos. Também permitindo visualizar o resultado das execuções. |
| **Atores** | | Especialista |
| **Pré-condições** | | Possuir as informações de conexão ao *data store* e *head node.* |
| **Pós-condições** | | Usuário é conectado. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| netHPC | Apresenta a tela da . | |
| Usuário | Preenche as informações de conexão ao *data store* e *head node* e clica no botão **OK**. | |
| netHPC | Valida se todas as informações foram preenchidas, mostrando uma mensagem de erro caso não tenham sido. | |
| netHPC | Conecta no *data store* utilizando as bibliotecas de acesso a banco de dados e no *head node* através de uma conexão TCP/IP. Com isso as informações de conexão são validadas. Caso ocorra algum problema nessa conexão, uma mensagem de erro é mostrada ao usuário informando o dessa situação. | |
| netHPC | Utilizando as conexões previamente estabelecidas as informações dos algoritmos, execuções e os nós computacionais são carregados na console, permitindo com isso que os demais casos de uso possam ser executados. | |
| **Tela** | | |
| Img002.png  Figura - Tela para conexão ao netHPC  Fonte: George Luiz Bittencourt (2009) | | |

Quadro - Caso de uso CSU-01

Fonte: George Luiz Bittencourt (2009)

No os detalhes do caso de uso utilizado para incluir um novo algoritmo podem ser visualizados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-02 |
| **Nome** | | Incluir Algoritmo |
| **Objetivo** | | Incluir um novo algoritmo na base de dados do netHPC. |
| **Atores** | | Especialista |
| **Pré-condições** | | O algoritmo deve ter sido desenvolvido previamente e compilado para inclusão. |
| **Pós-condições** | | O algoritmo é cadastrado na base de dados. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| Usuário | Expande a árvore **netHPC** → **Algorithms**. Clica com o botão direito sobre o nó **Algoritms** e seleciona a opção **Add Algorithm**. | |
| netHPC | Requisita ao usuário selecionar o arquivo que contém o algoritmo. | |
| Usuário | Seleciona o arquivo DLL que contém o algoritmo. | |
| netHPC | Apresenta a tela da . | |
| Usuário | Informa o nome e a descrição do algoritmo e clica no botão **OK**. | |
| netHPC | Cria uma ocorrência da classe **Algorithm** e a adiciona na base de dados. | |
| **Tela** | | |
| Img003.png  Figura – Tela para inclusão de algoritmo  Fonte: George Luiz Bittencourt (2009) | | |

Quadro – Caso de uso CSU-02

Fonte: George Luiz Bittencourt (2009)

No os detalhes do caso de uso utilizado para excluir um algoritmo podem ser verificados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-03 |
| **Nome** | | Excluir Algoritmo |
| **Objetivo** | | Excluir um algoritmo da base de dados do netHPC. |
| **Atores** | | Especialista |
| **Pré-condições** | | O algoritmo deve estar previamente cadastrado e não pode estar sendo executado. |
| **Pós-condições** | | O algoritmo e execuções deste são excluídas da base de dados. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| Usuário | Expande a árvore **netHPC** → **Algorithms**. | |
| netHPC | Carrega a lista de algoritmos a partir do *data store* e lista os algoritmos na console. | |
| Usuário | Seleciona o algoritmo desejado, clica com o botão direito e seleciona a opção **Delete**. | |
| netHPC | Questiona se o usuário deseja realmente efetuar a exclusão do algoritmo. | |
| Usuário | Responde que **Sim**. | |
| netHPC | Exclui o algoritmo e as execuções desse da base de dados. | |

Quadro - Caso de uso CSU-03

Fonte: George Luiz Bittencourt

No os detalhes do caso de uso utilizado para modificar um algoritmo podem ser verificados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-04 |
| **Nome** | | Modificar Algoritmo |
| **Objetivo** | | Modificar os dados de um algoritmo existente. |
| **Atores** | | Especialista |
| **Pré-condições** | | Possuir as novas informações do algoritmo. |
| **Pós-condições** | | O algoritmo é atualizado. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| Usuário | Expande a árvore **netHPC** → **Algorithms**. | |
| netHPC | Carrega a lista de algoritmos a partir do *data store* e lista os algoritmos na console. | |
| Usuário | Seleciona o algoritmo desejado, clica com o botão direito e seleciona a opção **Properties**. | |
| netHPC | Mostra a tela da . | |
| Usuário | Efetua a alteração no nome, na descrição ou no algoritmo através do botão **Update**. | |
| netHPC | Atualiza os dados do algoritmo no *data store*. | |
| **Tela** | | |
| Figura - Edição de algoritmo  Fonte: George Luiz Bittencourt (2009) | | |

Quadro - Caso de uso CSU-04

Fonte: George Luiz Bittencourt

No os detalhes do caso de uso utilizado para iniciar uma execução são mostrados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-05 |
| **Nome** | | Iniciar Execução |
| **Objetivo** | | Iniciar a execução de um algoritmo distribuindo o processamento dos *work items* entre os *computing nodes*. |
| **Atores** | | Especialista |
| **Pré-condições** | | O algoritmo deve estar cadastrado e o especialista deve possuir os parâmetros para a execução. |
| **Pós-condições** | | A execução é iniciada e após o seu término os resultados podem ser consultados através do relatório da execução. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| Usuário | Expande a árvore **netHPC** → **Algorithms**. Clica com o botão direito no algoritmo desejado e seleciona a opção **Start**. | |
| netHPC | Apresenta a tela da | |
| Usuário | Preenche o **nome** e a **descrição** da execução e clica na aba **Algorithm Parameters**. | |
| netHPC | Apresenta a tela da . A tela apresentada aqui implementa a interface **IConfigurationDialog** apresentada anteriormente. | |
| Usuário | Informa os parâmetros da execução de acordo com o algoritmo selecionado e clica na aba **Nodes**. | |
| netHPC | Apresenta a tela da . Nessa lista são listados somente os *computing nodes* disponíveis no momento. | |
| Usuário | Escolhe os *computing nodes* onde o algoritmo deve ser executado e clica na aba **Summary**. | |
| netHPC | Apresenta a tela da | |
| Usuário | Valida os dados da execução e clica no botão **Start**. | |
| netHPC | Cria no *data store* o registro referente a execução, inserindo os parâmetros da execução e relacionando os *computing nodes* que devem ser utilizados. Altera também o estado da execução para **Executing**. | |
| netHPC | Efetua a conexão ao *head node* enviando o comando para inicializar a execução. | |
| netHPC | O *head node* conecta em cada um dos *computing nodes* selecionados requisitando o início da execução. | |
| netHPC | Cada computing node efetua o download do algoritmo a partir do data store e inicia a execução do algoritmo. Ao finalizar o processamento do algoritmo a execução é marcada como **Completed**. | |
| Figura - Parametrização da execução  Fonte: George Luiz Bittencourt (2009) | | |
| Figura - Tela específica de parametrização da execução  Fonte: George Luiz Bittencourt (2009) | | |
| Figura - Escolha dos *computing nodes*  Fonte: George Luiz Bittencourt (2009) | | |
| Figura - Tela com o resumo da execução  Fonte: George Luiz Bittencourt (2009) | | |

Quadro - Caso de uso CSU-05

Fonte: George Luiz Bittencourt (2009)

No os detalhes do caso de uso utilizado para parar uma execução em andamento são mostrados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-06 |
| **Nome** | | Parar Execução |
| **Objetivo** | | Parar uma execução em andamento no estado em que se encontra. |
| **Atores** | | Especialista |
| **Pré-condições** | | O algoritmo deve estar sendo executado. |
| **Pós-condições** | | A execução do algoritmo é interrompida. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| Usuário | Expande **netHPC** → **Algorithms**. | |
| netHPC | Carrega todos os algoritmos cadastrados no *data store*. | |
| Usuário | Clica no algoritmo desejado. | |
| netHPC | Todas as execuções do algoritmo selecionado são listadas. | |
| Usuário | Seleciona a execução desejada onde o **Status** se encontra como **Executing**. | |
| Usuário | Clica com o botão direito e seleciona a opção **Cancel Execution**. | |
| netHPC | Pergunta se o usuário deseja realmente parar a execução selecionada. | |
| Usuário | Confirma o cancelamento. | |
| netHPC | Envia um mensagem para o *head node* requisitando que a execução seja abortada. | |
| netHPC | O *head node* recebe a solicitação e solicita a cada *computing node* que aborte a execução do algoritmo imediatamente. | |
| netHPC | Quando todos os computing nodes são parados o **Status** da execução é configurado como **Cancelled** nodata store. | |

Quadro - Caso de uso CSU-06

Fonte: George Luiz Bittencourt (2009)

No são mostrados os detalhes do caso de uso utilizado para verificar o resultado de uma execução distribuída.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-07 |
| **Nome** | | Visualizar Resultado da Execução |
| **Objetivo** | | Permitir ao especialista visualizar o resultado da execução do algoritmo. |
| **Atores** | | Especialista |
| **Pré-condições** | | O algoritmo deve ter sido executado. |
| **Pós-condições** | | O especialista tem acesso ao resultado da execução. |
| **Ativação** | | Usuário executa a console de gerenciamento do netHPC. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| Usuário | Expande **netHPC** → **Algorithms**. | |
| netHPC | Carrega todos os algoritmos cadastrados no *data store*. | |
| Usuário | Clica no algoritmo desejado. | |
| netHPC | Todas as execuções existentes são listadas. | |
| Usuário | Usuário seleciona a execução desejada e clica no botão **Visualizar Relatório**. | |
| netHPC | Gera o relatório para que o especialista possa verificar o resultado da execução. O relatório é gerado conforme o estilo da . | |
| **Tela** | | |
| Figura - Relatório de Exemplo  Fonte: George Luiz Bittencourt (2009) | | |

Quadro - Caso de uso CSU-07

Fonte: George Luiz Bittencourt (2009)

No os detalhes das ações executadas sempre que o agente é iniciado podem ser visualizados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-08 |
| **Nome** | | Iniciar Agente |
| **Objetivo** | | Adicionar um novo computador a grade de computadores, aumentado assim o poder computacional da mesma ou parar o agente. |
| **Atores** | | Windows |
| **Pré-condições** | | Acesso ao *data store* e ao *head node.* |
| **Pós-condições** | | O computador é adicionado a grade de computadores. |
| **Ativação** | | Sempre que o agente é iniciado. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| Windows | O sistema operacional inicia o agente. | |
| netHPC | Atualiza as informações do computador na grade de computadores criando ou atualizando o objeto do mesmo na classe **Node** e entra no estado onde fica aguardando solicitações. | |

Quadro - Caso de uso CSU-08

Fonte: George Luiz Bittencourt (2009)

No os detalhes do caso de uso utilizado para parar o agente podem ser verificados.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-09 |
| **Nome** | | Parar Agente |
| **Objetivo** | | Parar o agente interrompendo qualquer execução em andamento. |
| **Atores** | | Windows |
| **Pré-condições** | | O agente deve estar sendo executado. |
| **Pós-condições** | | O agente é parado. |
| **Ativação** | | Através da ferramenta de gerenciamento de serviços do Windows. |
| **Fluxo Normal** | | |
| Windows | O sistema operacional requisita a parada do agente | |
| netHPC | Caso o agente esteja no meio de uma execução o método **Abort** do algoritmo é executado. Caso contrário o mesmo é parado no mesmo momento. | |

Quadro - Caso de uso CSU-09

Fonte: George Luiz Bittencourt (2009)

No os detalhes do caso de uso utilizado para atualizar automaticamente o agente pode ser verificado.

|  |  |  |
| --- | --- | --- |
| **Identificador** | | CSU-10 |
| **Nome** | | Atualizar Agente |
| **Objetivo** | | Atualizar o agente com a última versão disponível no nó central. |
| **Atores** | | Relógio |
| **Pré-condições** | | Acesso ao *head node*. |
| **Pós-condições** | | Havendo uma atualização o agente é atualizado, caso contrário a versão permanece a mesma. |
| **Ativação** | | Automaticamente a cada 30 minutos. |
| **Fluxo Normal** | | |
| **Foco** | **Ação** | |
| netHPC | Envia uma mensagem para o *head node* requisitando a última versão do agente disponível. | |
| netHPC | Retorna uma mensagem com a última versão disponível. | |
| netHPC | Verifica se a versão retornada é maior que a versão atual. | |
| netHPC | Caso a versão instalada seja menor que a versão retornada a última versão é copiada a partir do nó central e instalada. Caso contrário nada é feito e o agente espera até a próxima verificação | |

Quadro - Caso de uso CSU-10

Fonte: George Luiz Bittencourt (2009)

#### Modelo de Estados

O diagrama da mostra os estados pelo qual o agente instalado nos *computing nodes* pode transitar enquanto estiver ativo. Os estados destacados com fundo preto são estados onde código do netHPC é executado. No estado com fundo branco é executado código específico do algoritmo.

Uma vez o agente iniciado pelo ator **ATOR-02** ele entra no estado **Aguardando execução** ondefica esperando ser acionado pelo *head node* a fim de executar algum algoritmo.

Quando uma execução é solicitada o *head node* conecta no agente através da rede e inicia a execução do algoritmo. Nesse momento o agente descarrega o algoritmo a partir do *data store* e o carrega em memória. *Threads* são criadas para cada unidade de execução presente no computador e o processamento inicia causando a transição do agente para o estado **Executando algoritmo**.

Durante a execução do algoritmo *work items* são necessários. Para obter um *work item* o algoritmo executa o método **GetWorkItem** que se encontra na classe base **netHPC.SDK.AlgorithmBase**. Nesse momento o agente muda seu estado para o estado **Buscando WorkItem para processar** e se comunica com o *head node*. Com o retorno desse método um *work item* é retornado e o agente retorna para o estado **Executando algoritmo**. Caso não existam *work items* disponíveis a execução é finalizada e o agente retorna para o estado **Aguardando execução** onde fica até a próxima requisição.

Após o *work item* ser processado o seu resultado deve ser reportado para que o especialista possa visualizar o mesmo através do relatório da execução. Esse reporte é feito através do método **ReportWorkItemResult**, que também se encontra na classe base **netHPC.SDK.AlgorithmBase**,passando como parâmetro um texto opcional que é armazenado no *data store*. Após o reporte do resultado o agente retorna para o estado **Executando algoritmo**.

Ainda durante a execução do algoritmo eventos **internos** do netHPC são gerados para auxiliar na análise do desempenho do mesmo. Essa mesma infra-estrutura de eventos pode ainda ser utilizada pelo algoritmo através de eventos **externos**. Para utilizar estes eventos o código inicialmente executa o método **MarkEventStart**,que se encontra na classe base **netHPC.SDK.AlgorithmBase**,passando como parâmetro o nome do evento a fim de marcar o início das atividades que se deseja mensurar. Nesse método o netHPC armazena em memória a data e hora do início do evento.

Após a finalização da atividade que se deseja mensurar o método **MarkEventEnd** deve ser executado passando como parâmetro o mesmo nome utilizado anteriormente no método **MarkEventStart**. Com isso o estado do agente é alterado para **Reportando evento interno ou externo** e nesse estado o netHPC calcula o tempo passado entre as duas chamadas e armazena o resultado no *data store*. Esse tempo tem precisão de milisegundos e pode ser visualizado no relatório de execução pelo especialista. O netHPC possui diversos eventos internos em seu código que são utilizados para medir o *overhead* do *middleware* no tempo total de execução. Após a inclusão do evento no *data store* o agente retorna ao estado **Executando algoritmo**.

A transição do agente para o estado **Aguardando execução** ocorre quando todos os *work items* foram processados ou quando a execução é interrompida a partir do *head node*. Quando isso ocorre o agente volta a aguardar uma nova execução.

O último estado pelo qual o agente pode transitar é o **Atualizando agente** e ocorre automaticamente a cada **30 minutos**, quando o **ATOR-03** o aciona.Esse estado somente é acionado quando o agente se encontra no estado **Aguardando execução**. Durante esse estado o agente se comunica com o *head node* a fim de verificar se existe alguma atualização do software agente. Existindo alguma atualização o agente se atualiza automaticamente e é reiniciado voltando para o estado **Aguardando execução**. Caso não exista nenhuma atualização disponível, o agente retorna ao estado **Aguardando Execução** e aguarda até a próxima verificação.

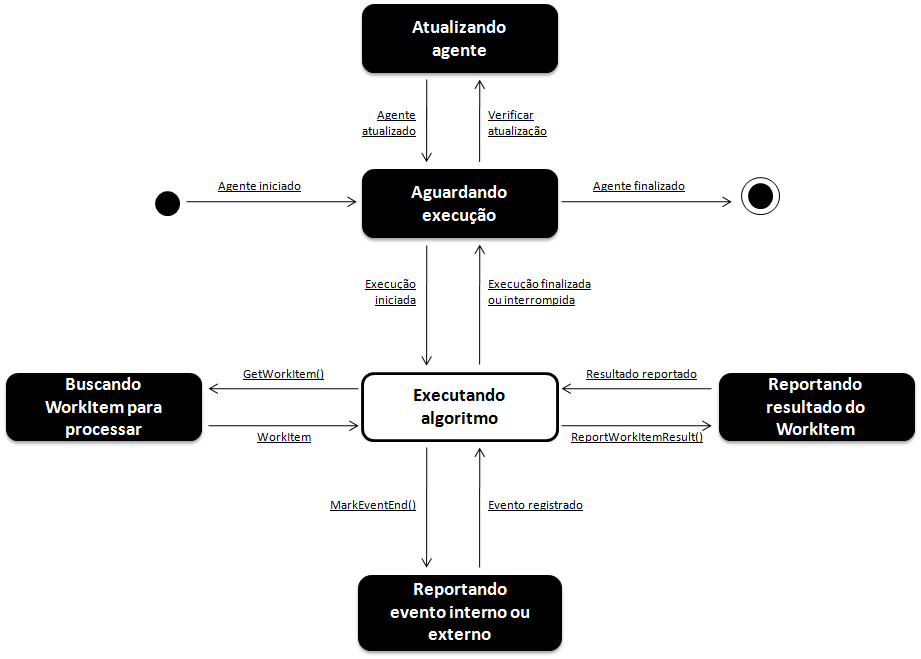


Figura - Diagrama de estados do agente nos computing nodes

Fonte: George Luiz Bittencourt (2009)

Além do agente instalado nos *computing nodes* existe o agente instalado no *head node* que também transita entre estados durante a execução de um algoritmo. A mostra os estados pelo qual esse agente podepassar enquanto estiver sendo executado.

Quando esse agente é iniciado ele passa para o estado **Aguardando execução** onde fica aguardando uma solicitação de execução vinda a partir da console de gerenciamento.

Quando uma requisição é efetuada o agente passa para o estado **Inicializando computing nodes**. Nesse estado o agente se comunica com cada um dos **computing nodes** selecionados requisitando que os mesmos iniciem a execução do algoritmo. Após a inicialização dos *computing nodes* o agente transita para o estado **Aguardando solicitação de WorkItem**.

Quando um *computing node* solicita um *work item* o agente passa para o estado **Gerando WorkItem** e executa o método **GetWorkItem** implementado pelo algoritmo e definido na interface **netHPC.SDK.IWorkBreaker**. Após esse método retornar um *work item* o mesmo é enviado para o *computing node* e o agente retorna ao estado **Aguardando solicitação de WorkItem.**

Quando não houverem mais *work items* para processar, a execução do algoritmo é considerada finalizada e o agente retorna para o estado **Aguardando execução** onde fica até a próxima solicitação**.**

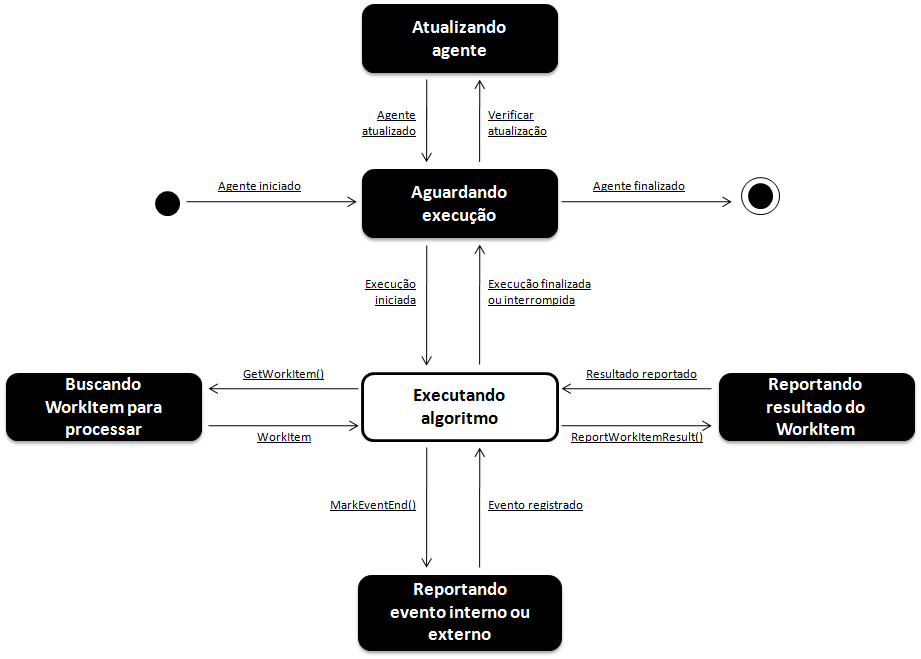


Figura - Diagrama de estados do agente no head node

Fonte: George Luiz Bittencourt (2009)

Já quando o especialista aborta a execução a partir da console de gerenciamento, o agente transita para o estado **Abortando computing nodes**. Nesse estado o agente se comunica com cada um dos *computing node* solicitando que o mesmo encerre a execução do algoritmo no ponto onde se encontra.

Após os *computing nodes* serem notificados do cancelamento da execução o agente volta ao seu estado inicial de **Aguardando execução** onde fica até a próxima requisição de execução.

#### Requisitos Não Funcionais

Os requisitos não funcionais na sequência foram definidos para o netHPC. No é possível verificar o requisito não funcional que define em que plataforma a console de gerenciamento deve ser construída.

|  |  |
| --- | --- |
| **Identificador** | REQNF-01 |
| **Nome** | A console de gerenciamento deve ser desenvolvida utilizando a Microsoft Management Console |
| **Definição** | A console de gerenciamento deve ser criada para ser executada sobre a Microsoft Management Console, que já se encontra instalada no Windows. |

Quadro - Requisito não funcional REQNF-01

Fonte: George Luiz Bittencourt (2009)

O mostra o requisito não funcional que define em que base de dados os dados devem ser armazenados.

|  |  |
| --- | --- |
| **Identificador** | REQNF-02 |
| **Nome** | Os dados devem ser armazenados em base de dados SQL Server |
| **Definição** | Todos os dados gerados devem ser armazenados no SGBD SQL Server 2008. |

Quadro - Requisito não funcional REQNF-02

Fonte: George Luiz Bittencourt (2009)

Já o mostra o requisito não funcional que determina que o agente deve ser um serviço do Windows.

|  |  |
| --- | --- |
| **Identificador** | REQNF-03 |
| **Nome** | O agente que será executado nos computadores deve ser executado mesmo quando nenhum usuário estiver utilizando o computador |
| **Definição** | O agente deve ser construído como um serviço do Windows para possibilitar sua execução mesmo quando ninguém estiver utilizando o computador. |

Quadro - Requisito não funcional REQNF-03

Fonte: George Luiz Bittencourt (2009)

### Análise dos Requisitos

Ao analisar os requisitos funcionais e não funcionais e também o caráter experimental do trabalho, foi decido pelo acadêmico não dividir o netHPC em subsistemas nesse momento, afinal o tamanho do sistema é ainda é pequeno, porém sua complexidade é grande.

Como nesse momento existe somente um ator definido, o especialista, não é efetuado nenhum controle de acesso aos casos de uso. Entretanto com a evolução do produto e o surgimento de novos atores, um mecanismo para controlar o acesso será implementado.

#### Arquitetura do Sistema

Na é possível visualizar a arquitetura da solução netHPC. Essa figura mostra alguns *computing nodes*, mostra ainda o *head node*, o *data store* e finalmente um computador com a console de gerenciamento a partir de onde o especialista pode controlar a distribuição das execuções.

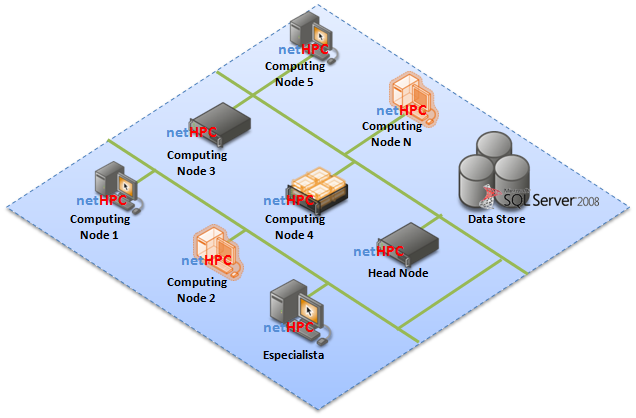


Figura - Arquitetura do netHPC

Fonte: George Luiz Bittencourt (2009)

Todos os computadores estão interligados através de uma rede local e nos *computing nodes* o agente está sendo executando em modo de **computação** e no *head node* esse mesmo agente está sendo executado em modo de **distribuição**.

### Projeto do Sistema

Na fase de projeto do sistema, o layout do relatório de execução da foi definido. Através desse relatório o especialista pode verificar como a execução do algoritmo foi dividida entre os computing nodes, visualizando o overhead do *middleware*, o número de *work items* gerados, os computadores que mais processarem *work items* e outras informações.

É importante frisar que o resultado propriamente dito da execução, o conteúdo do atributo **Text** da entidade **WorkItem**,deve ser extraído pelo especialista diretamente do banco de dados. Isso é necessário, pois o middleware deve ser genérico para qualquer tipo de algoritmo e, portanto não possui uma lógica para interpretar o conteúdo desse campo.

Todos os relatórios utilizados pelo netHPC foram desenvolvidos utilizando o componente **Reporting Services** do **SQL Server 2008**. Esse componente permite definir o layout e as consultas utilizadas para extrair os dados a partir da base dados. Uma vez o relatório gerado esse pode ser exibido na web e o seu conteúdo pode ser exportado em outros formatos, como Excel, PDF, CSV e outros.

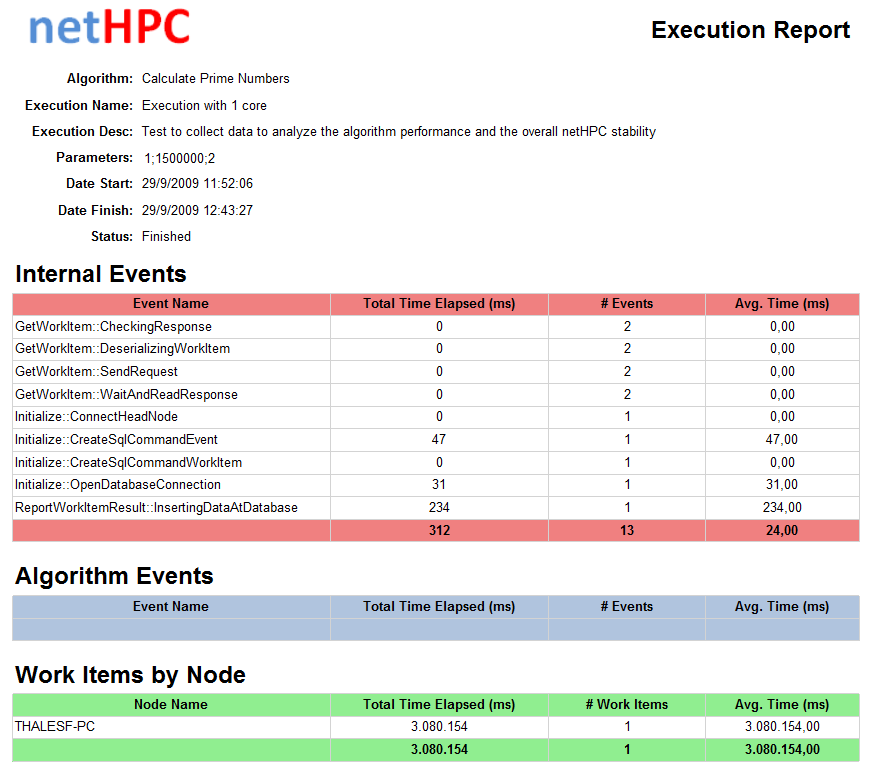


Figura - Layout do relatório de execução

Fonte: George Luiz Bittencourt (2009)

No layout da é possível ainda visualizar os eventos internos criados pela classe base **AlgorithmBase** a fim de mensurar o *overhead* causado pelo *middleware*. O detalhamento do propósito desses eventos se encontra na sequência.

* **GetWorkItem::CheckingResponse**: é o tempo utilizado para validar o retorno de uma requisição de um *work item*;
* **GetWorkItem::DeserializingWorkItem**: mede o tempo despendido no processo de gerar um objeto para representar o work item no algoritmo;
* **GetWorkItem::SendRequest**: representa o tempo utilizado para enviar a requisição de um *work item* para o *head node*;
* **GetWorkItem::WaitAndReadResponse**: é o tempo que o *computing node* fica esperando por um *work item* e também para ler a resposta a partir da rede;
* **Initialize::ConnectHeadNode**: mede o tempo utilizado para estabelecer a conexão TCP/IP com o *head node*;
* **Initialize::CreateSqlCommandEvent**: é o tempo utilizado para criar o objeto utilizado para inserir eventos no *data store*;
* **Initialize::CreateSqlCommandWorkItem**: representa o tempo utilizado para criar o objeto utilizado para inserir *work items* no *data store*;
* **Initialize::OpenDatabaseConnection**: mede o tempo utilizado para conectar no *data store* onde o resultado dos *work items* e eventos são inseridos;
* **ReportWorkItemResult::InsertingDataAtDatabase**: mensura o tempo utilizado para inserir o resultado do processamento de um *work item* no *data store*.

Todas as comunicações ocorrem sobre a pilha TCP/IP do sistema operacional Windows. Dessa forma eventuais problemas de desempenho nessa impactam diretamente no tempo total de processamento, entretanto esses problemas estão fora do controle do netHPC.

Quanto as telas utilizadas pelo netHPC, a mostra o layout de uma tela, que é utilizada para cadastrar um algoritmo na base de dados.

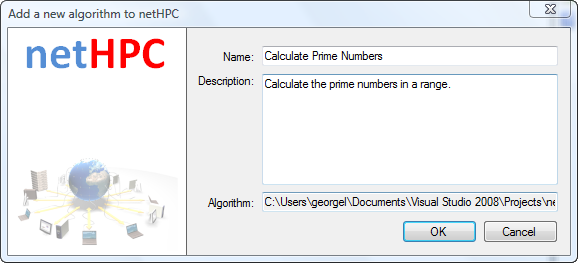


Figura - Tela de cadastro de algoritmo

Fonte: George Luiz Bittencourt (2009)

Já na tela da é possível visualizar a tela utilizada para iniciar a execução distribuída de um algoritmo.

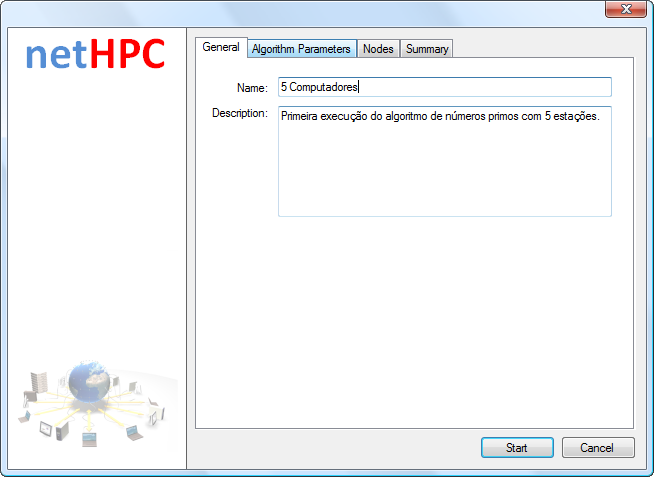


Figura - Tela utilizada para iniciar uma execução

Fonte: George Luiz Bittencourt (2009)

O sistema possui algumas outras telas, porém todas elas seguem o mesmo layout, mudando somente o seu objetivo no sistema. Todas essas telas foram desenvolvidas utilizando os controles gráficos providos pelo próprio .NET.

#### Projeto do Banco de Dados

Após a definição do diagrama de classes da o mesmo foi convertido para o modelo de entidades e relacionamentos da . Isso foi efetuado, pois os dados do netHPC serão persistidos em uma base de dados relacional. Para o armazenamento dos dados foi escolhido o SGBD **SQL Server 2008**.

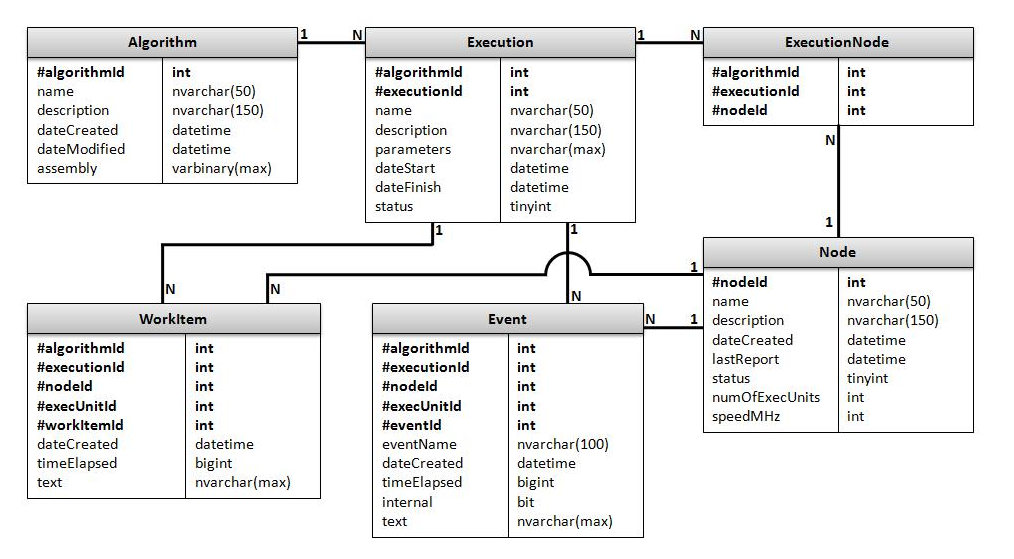


Figura - Modelo de entidades e relacionamentos

Fonte: George Luiz Bittencourt (2009)

Para efetuar a persistência dos dados no banco de dados relacional SQL Server foi utilizada a tecnologia **Entity Framework**, da Microsoft.Essa tecnologia permite definir um mapeamento entre as classes existentes no modelo orientado a objetos e as tabelas existentes na parte relacional do sistema. Isso facilita o desenvolvimento, pois é necessário definir os comandos SQL no código, esse framework gera esses comandos em tempo de execução

Na é possível visualizar o diagrama gerado automaticamente pela ferramenta. Cada entidade no modelo relacional tornou-se uma classe que pode ser utilizada em código .NET para persistir o registro de forma transparente. Essas classes foram geradas no pacote **netHPC.Shared** que, conforme já mostrado, contém as entidades utilizadas pelo netHPC.

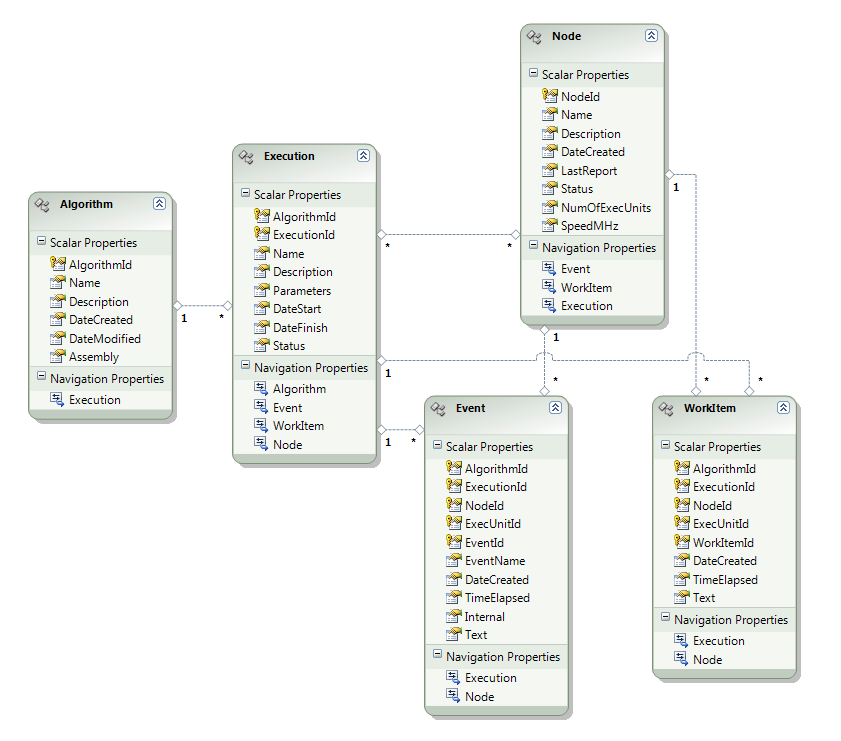


Figura - Diagrama utilizado pelo Entity Framework

Fonte: George Luiz Bittencourt (2009)

#### Definição do Dicionário de Dados

A partir do modelo de entidades e relacionamentos da o dicionário de dados foi elaborado. Esse dicionário pode ser visualizado na sequência.

No é possível verificar os detalhes da entidade **Algorithm**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | | Algorithm | |
| **Objetivo** | | Armazenar os algoritmos disponíveis no netHPC para computação distribuída | |
| **Atributos** | | | |
| **Nome** | **Tipo** | | **Descrição** |
| **algorithmId** | **int** | | Chave primária da tabela e também código do algoritmo |
| name | nvarchar(50) | | Nome do algoritmo |
| description | nvarchar(150) | | Descrição opcional do algoritmo |
| dateCreated | datetime | | Data e hora em que o algoritmo foi cadastrado |
| dateModified | datetime | | Date e hora da última modificação do algoritmo |
| assembly | varbinary(max) | | Código compilado do algoritmo em formato executável pelo .NET |

Quadro - Tabela Algorithm

Fonte: George Luiz Bittencourt (2009)

No é possível verificar os detalhes da entidade **Execution**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | | Execution | |
| **Objetivo** | | Armazenar o histórico e os resultados das execuções dos algoritmos para posterior análise pelo especialista | |
| **Atributos** | | | |
| **Nome** | **Tipo** | | **Descrição** |
| **algorithmId** | **int** | | Chave primária e estrangeira referenciando o algoritmo ao qual a execução pertence. |
| **executionId** | **int** | | Chave primária da tabela e também código da execução. Esse código é sequencial dentro do algoritmo. |
| name | nvarchar(50) | | Nome atribuído a execução pelo especialista ao iniciar a mesma. |
| description | nvarchar(150) | | Descrição opcional atribuída a execução pelo especialista ao iniciar a mesma. |
| parameters | nvarchar(max) | | Parâmetros retornados pelo método **GetParameters** da interface **netHPC.SDK. IConfigurationDialog**. |
| dateStart | datetime | | Date e hora em que a execução foi iniciada. |
| dateFinish | datetime | | Date e hora em que a execução foi finalizada. Esse atributo somente é definido quanto o atributo status for diferente de **0**. |
| status | tinyint | | Estado da execução do algoritmo, onde:  0 – execução em andamento  1 – execução finalizada com sucesso  2 – execução cancelada pelo especialista |

Quadro - Tabela Execution

Fonte: George Luiz Bittencourt (2009)

No é possível verificar os detalhes da entidade **Node**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | | Node | |
| **Objetivo** | | Armazenar os nós que fazem parte da grade computacional. | |
| **Atributos** | | | |
| **Nome** | **Tipo** | | **Descrição** |
| **nodeId** | **int** | | Chave primária do nó. |
| name | nvarchar(50) | | Nome do nó. Esse é o nome do computador retornado pelo próprio Windows através de suas APIs |
| description | nvarchar(150) | | Descrição opcional do nó. |
| dateCreated | datetime | | Data e hora em que o nó se juntou a grade computacional. |
| lastReport | datetime | | Data e hora da última vez que o agente do nó se reportou a grade computacional. |
| status | tinyint | | Estado do nó, onde:  0 – Ocioso;  1 – Processando Algoritmo;  2 – Desconhecido; |
| numOfExecUnits | int | | Número de unidades de execução. A seguinte fórmula é usada para calcular esse valor:  **Nr. Proc. Físicos X Nr. Cores por Proc. Físico** |
| speedMHz | int | | Velocidade em MHz das unidades de execução. |

Quadro - Tabela Node

Fonte: George Luiz Bittencourt (2009)

No é possível verificar os detalhes da entidade **ExecutionNode**. Essa entidade surgiu devido à existência de uma associação **muitos para muitos** existente entre as entidades **Execution** e **Node**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | | ExecutionNode | |
| **Objetivo** | | Relacionar a execução de um algoritmo com os nós utilizados no processamento dessa. | |
| **Atributos** | | | |
| **Nome** | **Tipo** | | **Descrição** |
| **algoritmId** | **int** | | Chave primária e estrangeira para a tabela **Execution**. |
| **executionId** | **int** | | Chave primária e estrangeira para a tabela **Execution**. |
| **nodeId** | **int** | | Chave primária e estrangeira para a tabela **Node**. |

Quadro - Tabela ExecutionNode

Fonte: George Luiz Bittencourt (2009)

No é possível verificar os detalhes da entidade **WorkItem**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | | WorkItem | |
| **Objetivo** | | Armazenar o resultado dos itens de trabalho processados pelos nós. | |
| **Atributos** | | | |
| **Nome** | **Tipo** | | **Descrição** |
| **algorithmId** | **int** | | Chave primária e estrangeira para a tabela **Execution**. |
| **executionId** | **int** | | Chave primária e estrangeira para a tabela **Execution**. |
| **nodeId** | **int** | | Chave primária e estrangeira para a tabela **Node**. |
| **execUnitId** | **int** | | Identificador da unidade de execução que processou o item de trabalho em questão. Esse identificador é relativo ao nó onde foi gerado. |
| **workItemId** | **int** | | Identificador do item de trabalho em questão. |
| dateCreated | datetime | | Data e hora em que o resultado do item de trabalho foi finalizado |
| timeElapsed | bigint | | Tempo total em **milisegundos** para processar o item de trabalho |
| text | nvarchar(max) | | Texto opcional com o resultado do item de trabalho |

Quadro - Tabela WorkItem

Fonte: George Luiz Bittencourt (2009)

No é possível verificar os detalhes da entidade **Event**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | | Event | |
| **Objetivo** | | Armazenar a duração de eventos internos do netHPC e externos, onde esses últimos são definidos pelo algoritmo. | |
| **Atributos** | | | |
| **Nome** | **Tipo** | | **Descrição** |
| **algorithmId** | **int** | | Chave primária e estrangeira para a tabela **Execution**. |
| **executionId** | **int** | | Chave primária e estrangeira para a tabela **Execution**. |
| **nodeId** | **int** | | Chave estrangeira para a tabela **Node** indicando em que nó esse evento foi gerado |
| **execUnitId** | **int** | | Identificador da unidade de execução que processou o item de trabalho em questão. Esse identificador é relativo ao nó onde foi gerado. |
| **eventId** | **int** | | Identificador do evento em questão. |
| eventName | nvarchar(100) | | Nome do evento |
| dateCreated | datetime | | Data e hora em que o evento foi finalizado. |
| timeElapsed | bigint | | Tempo total em **milisegundos** para processar o evento. |
| internal | bit | | Indica se é um evento interno do netHPC ou um evento criado pelo algoritmo. |
| text | nvarchar(max) | | Texto opcional com detalhes sobre o evento |

Quadro - Tabela Event

Fonte: George Luiz Bittencourt (2009)

### Programação do Sistema

Todo o desenvolvimento do netHPC foi efetuado utilizando ferramentas da Microsoft. A programação foi efetuada utilizando a linguagem orientada a objetos **C# 3.0** e o ambiente de desenvolvimento utilizado foi o **Visual Studio Team System 2008**. Esse ambiente pode ser visualizado na .

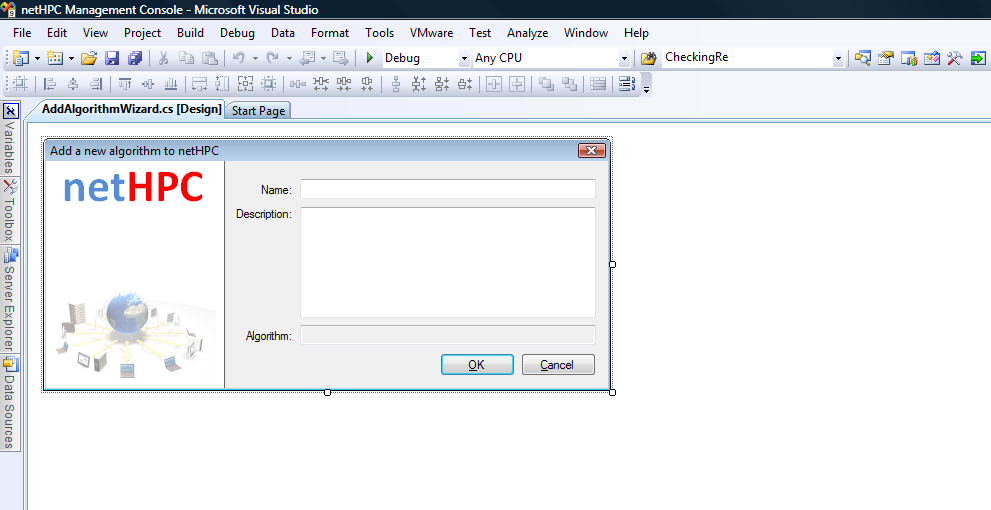


Figura - Ambiente de desenvolvimento utilizado

Fonte: George Luiz Bittencourt (2009)

O netHPC foi divido em vários projetos do Visual Studio a fim de agrupar código relacionado. Na é possível visualizar alguns desses projetos através da ferramenta Solution Explorer do Visual Studio.

O netHPC é composto dos seguinte projetos no Visual Studio:

* **netHPC.ManagementConsole**:é a console de gerenciamento utilizada pelos usuários;
* **netHPC.SDK**: contém o kit de desenvolvimento de software que o programador deve utilizar para implementar um algoritmo que possa ser executado sobre o netHPC;
* **netHPC.Shared**: contém as entidades utilizadas para armazenar e recuperar os dados na base de dados;
* **netHPC.Service**: é o agente que é executado tanto no *head node* quanto no *computing node*. Esse projeto gera como saída um serviço do Windows;
* **netHPC.Samples.PrimeNumbers**:é o projeto que contém o algoritmo de exemplo utilizado para calcular os números primos em uma faixa de números. Para o teste em campo foi utilizado esse algoritmo.

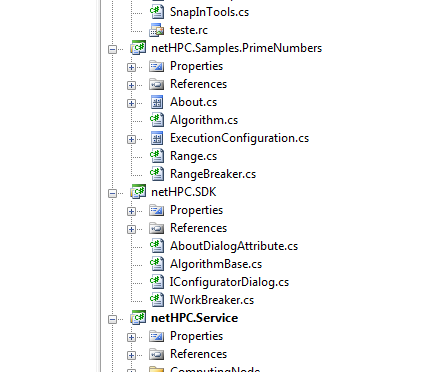


Figura - Parte dos projetos que compõem o netHPC no Visual Studio 2008

Fonte: George Luiz Bittencourt

A console de gerenciamento utilizada pelos usuários foi criada para ser utilizada sobre a Microsoft Management Console (MMC). Na MMC são utilizados **snap-ins** que representam software desenvolvidos para essa console. Na é possível visualizar o *snap-in* criado para o netHPC sendo incluído na MMC para utilização.

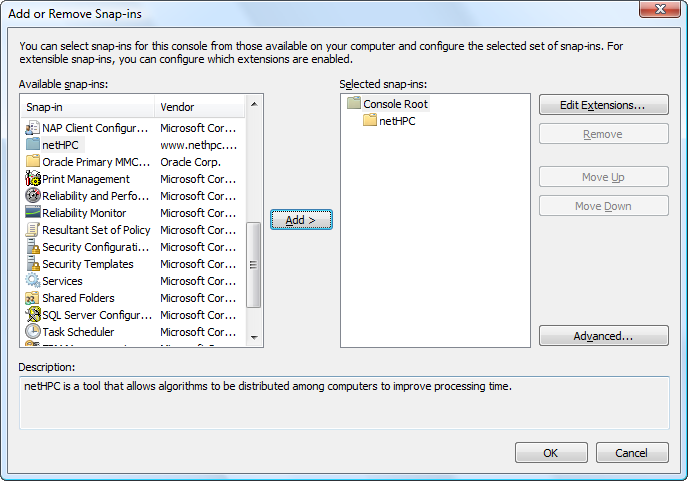


Figura - Inclusão de console de gerenciamento na MMC

Fonte: George Luiz Bittencourt (2009)

Após a inclusão do *snap-in* na MMC o mesmo pode ser visualizado na . Nessa console existe no lado esquerdo uma árvore de navegação composta por nós que o usuário deve utilizar para navegar pelo sistema. Clicando com o botão direito sobre um dos nós as tarefas vinculadas a ele são exibidas.

No painel central as informações referentes ao nó selecionado são mostradas. Na figura em questão é possível visualizar que o nó **Algorithms** está selecionado e no painel central o algoritmo **Calculate Prime Numbers,** quefoi utilizado nos testes em campo, é apresentado.

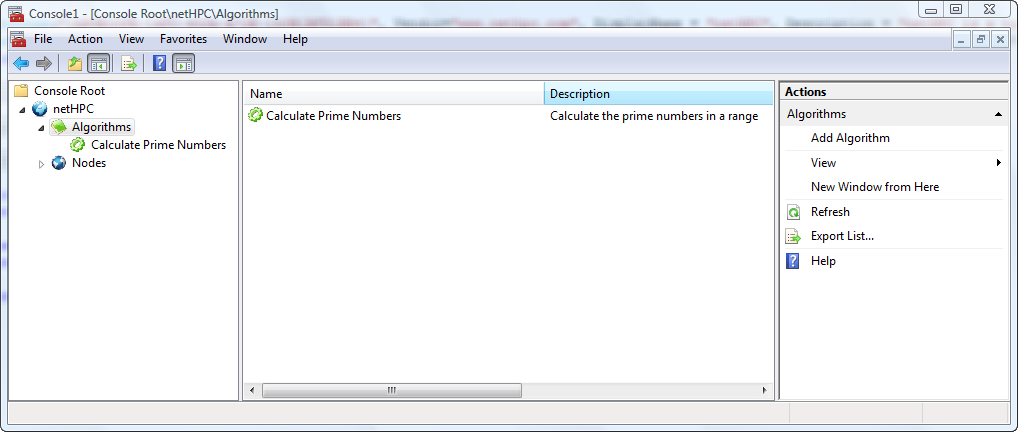


Figura - Console de gerenciamento criada sobre a MMC

Fonte: George Luiz Bittencourt (2009)

Na o serviços sobre os quais os algoritmos são executados, e que esta presente em cada *computing node*, pode ser visualizado. Esse serviço foi construído utilizando a interface definida pela Microsoft para esse tipo de aplicação e permite que o mesmo seja executado mesmo quando não usuário algum conectado no servidor e sua inicialização pode ser configurada para ser automático em conjunto com a inicialização do próprio Windows.

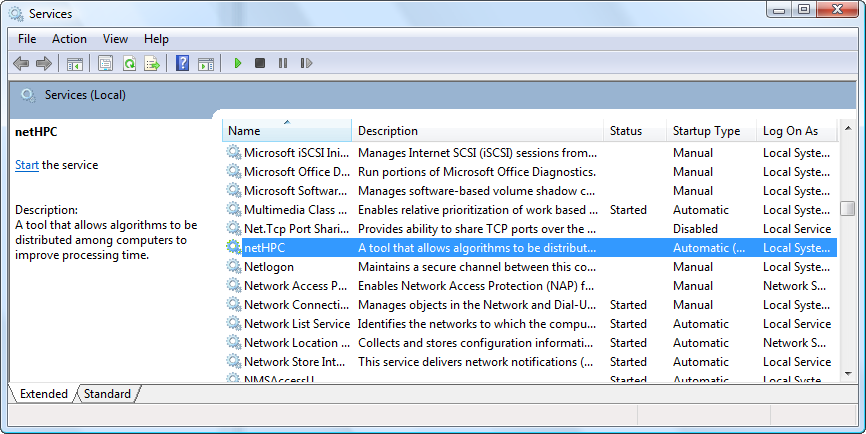


Figura - Serviço do netHPC criado no Windows

Fonte: George Luiz Bittencourt (2009)

### Teste em Campo do Sistema

Para validar a solução de HPC desenvolvida foi definido, em conjunto com o orientador específico, a construção de um algoritmo que possa comprovar os benefícios desse tipo de processamento e também o correto funcionamento do *middleware*.

O acadêmico escolheu desenvolver um algoritmo que calcule os números primos existentes em uma faixa de números fornecido. O detalhamento desse algoritmo e os resultados podem ser visualizados na sequência.

O teste de distribuição do algoritmo foi executado no dia **29/09/2009** pelo acadêmico nas dependências da Integral Informática Ltda. Esse teste consistiu em distribuir o processamento do cálculo dos números primos entre 1 e 1.500.000 entre 10 processadores. Os resultados desse teste podem ser verificados na sequência.

Na é possível visualizar as classes criadas para suportar esse algoritmo de testes. É importante frisar que esse algoritmo utiliza o kit de desenvolvimento de software criado para o netHPC já definido anteriormente.

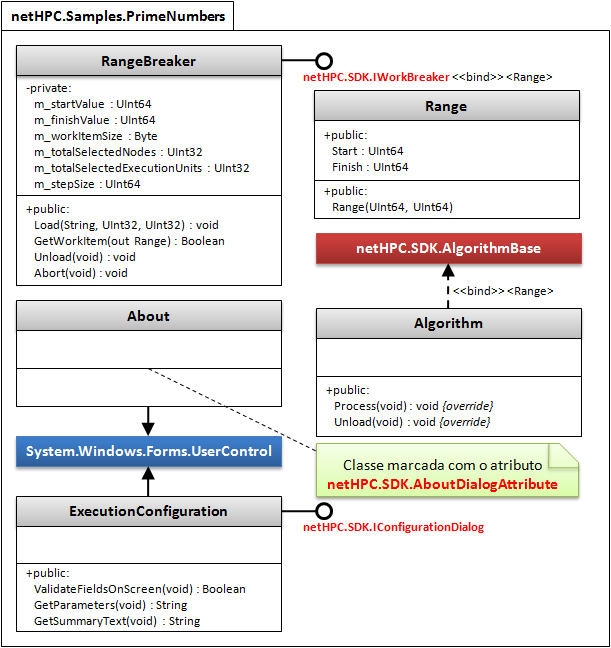


Figura - Diagrama de classes do algoritmo para cálculo de números primos

Fonte: George Luiz Bittencourt (2009)

A classe **Range** (faixa em português) representa o *work item* que o algoritmo irá processar. Essa classe define somente duas propriedades que são o número inicial e final da faixa de números. Com exceção do construtor essa classe não possui nenhum outro método, já que seu único objetivo é transportar o *work item* entre o *head node* e os *computing nodes*.

A classe **RangeBreaker** (quebrador de faixa em português) implementa a interface **netHPC.SDK.IWorkBreaker** e é executada no *head node*. Essa classe quebra a faixa total de números em faixas menores (*work items*) que são distribuídos para os *computing nodes* através da classe **Range**.Basicamente o *head node* executa o método **GetWorkItem** quantas vezes forem necessário até o mesmo não retornar nenhum *work item*. Quando não houverem mais *work items* é por que o trabalho total foi processado e a execução é finalizada.

A mostra o código utilizado para inicializar a classe. Essa lógica basicamente determina o valor da variável **m\_stepSize** que contém o tamanho da faixa de números de cada *work item*. Nessa versão inicial do algoritmo, apesar de existir uma forma de controlar o tamanho do *work item*, o acadêmico optou por gerar *work items* de tamanho pequeno, porém com a evolução do algoritmo um novo parâmetro será colocado na tela de configuração. Esse parâmetro permitirá o especialista controlar o tamanho médio dos *work items* e assim verificar alterações no *overhead* causado pelo netHPC na execução.

|  |
| --- |
| public void Load(string executionParameters, uint totalSelectedNodes, uint totalSelectedExecutionUnits)  {  String[] arrTmp = executionParameters.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);  m\_startValue = UInt64.Parse(arrTmp[0]) - 1;  m\_finishValue = UInt64.Parse(arrTmp[1]);  m\_workItemSize = (Byte)Math.Pow(3 \* (Byte.Parse(arrTmp[2]) + 1), Byte.Parse(arrTmp[2]));  m\_totalSelectedNodes = totalSelectedNodes;  m\_totalSelectedExecutionUnits = totalSelectedExecutionUnits;  m\_stepSize = (m\_finishValue - m\_startValue) / (m\_totalSelectedExecutionUnits \* m\_workItemSize);  } |

Figura - Método utilizado para inicializar o RangeBreaker

Fonte: George Luiz Bittencourt (2009)

A mostra o código que cria os *work items* conforme as solicitações dos *computing nodes* são recebidas pelo *head node*.

|  |
| --- |
| public Boolean GetWorkItem(out Range workItem)  {  workItem = null;  if ((m\_finish == true) || (m\_startValue >= m\_finishValue))  return false;    if (m\_totalSelectedExecutionUnits == 1)  {  workItem = new Range(m\_startValue, m\_finishValue);  m\_finish = true;  return true;  }  else  {  workItem = new Range(m\_startValue + 1, 0);  m\_startValue = (m\_startValue + 1) + m\_stepSize;  workItem.Finish = m\_startValue;  return true;  }  } |

Figura - Código fonte C# responsável por gerar os work items

Fonte: George Luiz Bittencourt (2009)

A classe **Algorithm**, o *cerne* do algoritmo, define o passos que serão executados para cada *work item* recebido.Essa classe herda a classe **netHPC.SDK.AlgorithmBase** e se utiliza dos métodos dessa para buscar *work items*, reportar o resultado do processamento e também criar eventos. O código utilizado para processar os *work items* e determinar se os números existentes no **Range** são primos pode ser visualizado na . Cada número primo encontrado é concatenado em um objeto **StringBuilder** e esse objeto é atualizado no *data store* como o resultado do *work item* ao término do processamento desse.

|  |
| --- |
| protected override void Process()  {  StringBuilder stringBuilder;  Range range = null;  Boolean prime = true;  while (GetWorkItem(out range))  {  stringBuilder = new StringBuilder(2000);  stringBuilder.AppendLine(" Start: " + range.Start.ToString());  stringBuilder.AppendLine("Finish: " + range.Finish.ToString());  for (UInt64 tmpCounter = range.Start; tmpCounter <= range.Finish; tmpCounter++)  {  if ((tmpCounter % 2 == 0) || (tmpCounter % 5 == 0))  continue;  prime = true;  for (UInt64 i = 3; i < tmpCounter; i++)  {  if (tmpCounter % i == 0)  {  prime = false;  break;  }  }  if (prime)  stringBuilder.AppendLine(tmpCounter.ToString());  }  ReportWorkItemResult(stringBuilder.ToString());  }  } |

Figura - Algoritmo utilizado para processar um work item

Fonte: George Luiz Bittencourt (2009)

A classe **About** herda a classe **System.Windows.Forms.UserControl** e é marcada com o atributo **netHPC.SDK.AboutDialogAttribute**. Essa classe simplesmente mostra um texto contendo os direitos autorais do algoritmo. Na é possível visualizar a tela criada para o algoritmo de testes. Essa tela pode ser visualizada através das propriedades do algoritmo na console de gerenciamento.

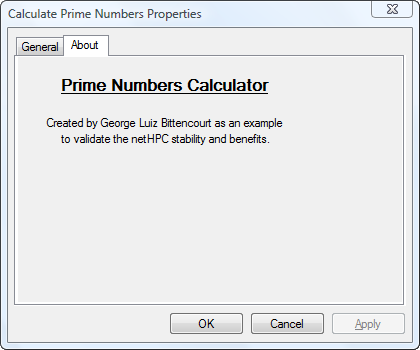


Figura - Tela com detalhes do autor do algoritmo

Fonte: George Luiz Bittencourt (2009)

A classe **ExecutionConfiguration** implementa a interface **netHPC.SDK.IConfigurationDialog** e também herda a classe **System.Windows.Forms.UserControl**. Na é possível visualizar essa tela mostrando os parâmetros utilizados no teste.

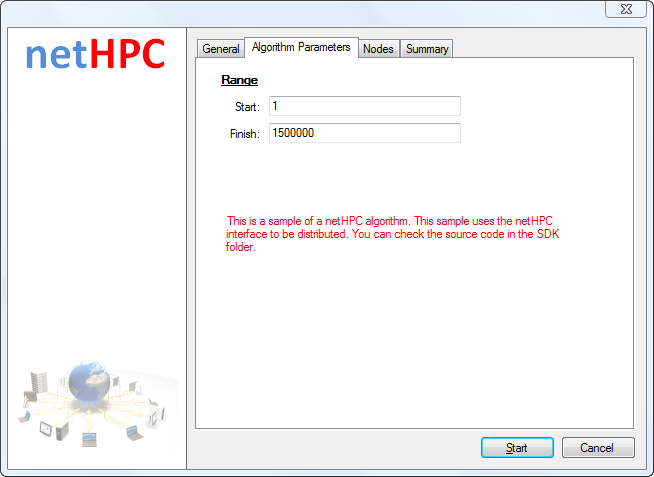


Figura - Tela com os parâmetros utilizados no teste

Fonte: George Luiz Bittencourt (2009)

Na é possível visualizar a última execução do teste onde oito computadores foram selecionados para processamento, totalizando dez processadores.

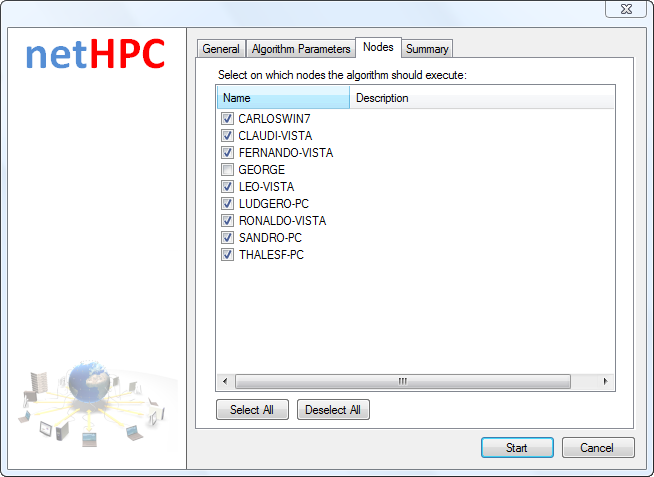


Figura - Tela de seleção dos computadores onde o algoritmo será executado

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | CARLOSWIN7 |
| **Processador** | Intel(R) Core(TM) 2 Duo CPU E4400 @ 2.00GHz |
| **Memória Cache** | 2 MB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuração do computador CARLOSWIN7

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | CLAUDI-VISTA |
| **Processador** | AMD Sempron(tm) Processor 3400+ |
| **Memória Cache** | 256 kB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows Vista |

Quadro - Configuração do computador CLAUDI-VISTA

Fonte: George Luiz Bittencourt

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | FERNANDO-VISTA |
| **Processador** | Intel(R) Core(TM) 2 Duo CPU E4400 @ 2.00GHz |
| **Memória Cache** | 2 MB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows Vista |

Quadro - Configuração do computador FERNANDO-VISTA

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *head node* durante o teste. Esse computador foi o responsável por decompor o trabalho total em unidades menores e distribuir as mesmas para os *computing nodes*. Foi nesse computador também que os dados foram armazenados no *data store*.

|  |  |
| --- | --- |
| **Computador** | GEORGE |
| **Processador** | AMD Sempron(tm) Processor 3400+ |
| **Memória Cache** | 256 kB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuração do computador GEORGE

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | LEO-VISTA |
| **Processador** | Intel(R) Pentium(R) 4 CPU 2.80GHz |
| **Memória Cache** | 1 MB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuraçao do computador LEO-VISTA

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | LUDGERO-PC |
| **Processador** | Intel(R) Celeron(R) CPU 2.40GHz |
| **Memória Cache** | 256 kB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuração do computador LUDGERO-PC

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | RONALDO-VISTA |
| **Processador** | AMD Sempron(tm) Processor 3400+ |
| **Memória Cache** | 256 kB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuração do computador RONALDO-VISTA

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | SANDRO-PC |
| **Processador** | Intel(R) Celeron(R) CPU 2.40GHz |
| **Memória Cache** | 256 kB |
| **Memória RAM** | 1 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuração do computador SANDRO-PC

Fonte: George Luiz Bittencourt (2009)

No é possível visualizar as configurações do computador utilizado como *computing node* durante o teste.

|  |  |
| --- | --- |
| **Computador** | THALESF-PC |
| **Processador** | AMD Sempron(tm) Processor 3400+ |
| **Memória Cache** | 256 kB |
| **Memória RAM** | 2 GB |
| **Sistema Operacional** | Windows 7 |

Quadro - Configuração do computador THALESF-PC

Fonte: George Luiz Bittencourt (2009)

O resultado das execuções pode ser visualizado na . Esses dados foram extraídos através dos relatórios de execução providos pelo netHPC. Esses relatórios podem ser visualizados no **APÊNDICE A – RELATÓRIOS DAS EXECUÇÕES DE TESTE** no final do trabalho.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Execução** | 1 | 2 | 3 | 4 | 5 | 6 |
| **Hora Inicial** | 11:52 | 13:07 | 13:24 | 18:04 | 18:16 | 18:31 |
| **Hora Final** | 12:43 | 13:23 | 13:36 | 18:13 | 18:24 | 18:35 |
| **Nr. Total de Cores** | 1 | 2 | 3 | 4 | 5 | 10 |
| **Nr. Total de Computadores** | 1 | 1 | 2 | 3 | 4 | 8 |
| **Duração da Execução** | 0:51:21 | 0:15:24 | 0:11:55 | 0:09:29 | 0:08:02 | 0:03:40 |
| **Nr. de Work Items Gerados e Distribuídos** | 1 | 162 | 243 | 324 | 405 | 810 |
| **Trabalho Total** | 0:51:20 | 0:30:32 | 0:35:20 | 0:37:25 | 0:39:37 | 0:35:27 |
| **Tempo no netHPC (segs)** | 0,31 | 1,65 | 2,30 | 7,99 | 5,10 | 20,56 |
| **% Overhead do netHPC** | 0,010 | 0,090 | 0,108 | 0,355 | 0,214 | 0,957 |

Tabela - Resultados dos testes de execução

Fonte: George Luiz Bittencourt (2009)

Ao analisar a duração da primeira execução (51 mins e 21 segs), com somente um *core,* contra a última execução (3 mins e 40 segs), com dez *cores*, percebe-se uma grande redução na duração do processamento. A última duração corresponde a **7%** da primeira execução, uma redução de mais de **90%** em relação a primeira execução.Com esse tempo é possível executar **quatorze** vezes o mesmo algoritmo em dez *cores* enquanto a execução em 1 *core* é executada.

Analisando também o número de *work items* gerados e distribuídos pelo processo de decomposição executado no *head node,* é possível verificar que o mesmo se adaptou bem ao maior número de cores, criando mais *work items* conforme o número de cores foi crescendo.

O *overhead* gerado pelo *middleware* e demais componentes foi mínimo. Com dez cores o *overhead* chegou a praticamente **1%** do tempo total. Quanto menor esse valor, melhor, pois indica que a maior parte do tempo da execução está sendo despendido no processamento do algoritmo.

Analisando isoladamente a primeira execução com a segunda, percebe-se que a qualidade do processador em questão influenciou muito na duração, afinal ambas as execuções contaram com um computador físico somente, porém o segundo tinha dois *cores* e o tempo (15 mins e 24 egs) foi bem menor que a metade da primeira execução (51 mins e 21 segs). O processador da primeira execução é um **AMD Sempron**, que é uma linha da AMD destinada a demandas não focadas em performance, já o segundo processador é um **Intel Core 2 Duo**, destinada a situações onde a performance é importante.

O gráfico da mostra que conforme a duração da execução foi diminuindo o trabalho foi aumentando. Esse comportamento é esperado, afinal a duração é o tempo total entre o início e o fim da execução, já o trabalho é a soma do tempo utilizado por cada *core* para processar os *work items*. Com isso quanto mais *cores* forem sendo adicionados a grade de computadores o trabalho irá aumentar, pois existirão mais *cores* trabalhando do processamento dos *work items* e a duração irá diminuir, pois existirão mais cores trabalhando em paralelo para diminuir esse tempo.

Figura - Gráfico da duração e trabalho por número de cores

Fonte: George Luiz Bittencourt (2009)

Já o gráfico da mostra o número de *work items* gerados e distribuídos para os *computiing nodes*. O comportamento mostrado no gráfico é o esperado, pois mais *cores* estão requisitando *work items* para processar e a classe **RangeBreaker** está quebrando a faixa de números em fatias menores.

Figura - Gráfico do número de Work Items por Número de Cores

Fonte: George Luiz Bittencourt (2009)

### Implantação do Sistema

O benefício da computação em grade está na quantidade e qualidade dos computadores disponíveis para processamento. Sob o ponto de vista da implantação isso traz um desafio para o administrador, pois o mesmo precisará instalar o agente do netHPC em vários computadores aumentando com isso o custo total do proprietário (TCO, do inglês total cost of ownership) da solução.

Como o netHPC executa sobre o sistema operacional Windows e praticamente todas as redes de universidades, faculdades e empresas possuem o serviço de diretório **Active Directory** da Microsoft, a implantação pode ser efetuada utilizando o recurso de instalação remota provida por este. Esse recurso permite instalar remotamente software em estações sem a necessidade de alguém se deslocar até a estação. Na é possível visualizar essa funcionalidade. Nessa figura pode ser visualizado o pacote de instalação do netHPC definido para ser instalado em todo o domínio Active Directory. Com isso quando cadê computador for iniciado o Windows irá instalar e iniciar o agente em cada computador.

O pacote de instalação do netHPC foi gerado utilizando a ferramenta de código fonte aberto **WiX** (Windows Installer XML). Essa ferramenta gera o instalador do sistema com base em um arquivo XML de configuração. O instalador é gerado no formato do **Microsoft Installer** (MSI), que é um formato de arquivo criado pela Microsoft e que permite efetuar instalação de forma remota e transacional no Windows.

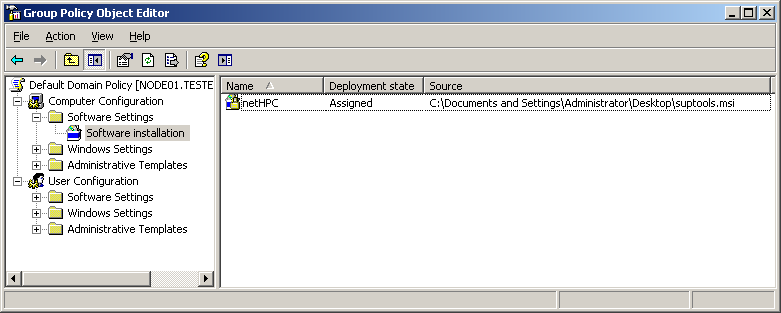


Figura - Instalação do netHPC através do Active Directory

Fonte: George Luiz Bittencourt (2009)

Além da instalação, o Active Directory pode ainda ser utilizado para efetuar a configuração do netHPC nos computadores através do recurso **Group Policy Object** (**GPO**). Com esse recurso é possível gravar dados no registro do Windows e estes podem ser lidos pelos aplicativos. O netHPC foi construído pensando nisso e todas as configurações de acesso utilizadas pelos *computing nodes* são extraídas a partir do registro do Windows. Na é possível verificar as configurações do netHPC sendo distribuídas pelo Active Directory.

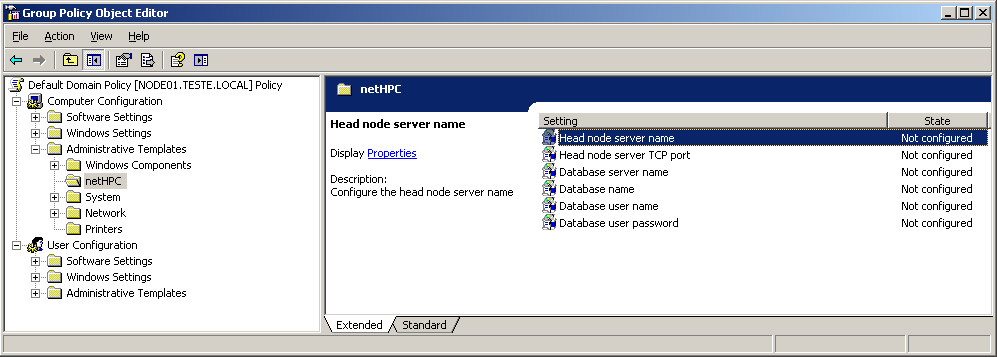


Figura - Configuração do netHPC através do Active Directory

Fonte: George Luiz Bittencourt (2009)

Independente do método de implantação utilizado, o netHPC possui os seguintes pré-requisitos de hardware e software:

* Computador com hardware compatível com o Windows XP ou superior;
* Windows XP ou superior;
* .NET Framework 3.5 SP1;
* Microsoft Management Console 3.0;
* Conectividade com o *head node* através de uma rede TCP/IP.

Para o *data store* é necessário o **SQL Server 2008**, podendo ser utilizado a versão **Express** do mesmo, que é gratuita.

### Documentação do Sistema

A documentação do kit de desenvolvimento de software foi construída utilizando a ferramenta de código fonte aberto **nDoc**. Essa ferramenta utiliza os comentários inseridos no código fonte, em formato XML, para gerar o arquivo de ajuda no formato utilizado pelo Windows.

Na é possível visualizar o arquivo de ajuda gerado pela ferramenta contendo a documentação do kit de desenvolvimento de software do netHPC.

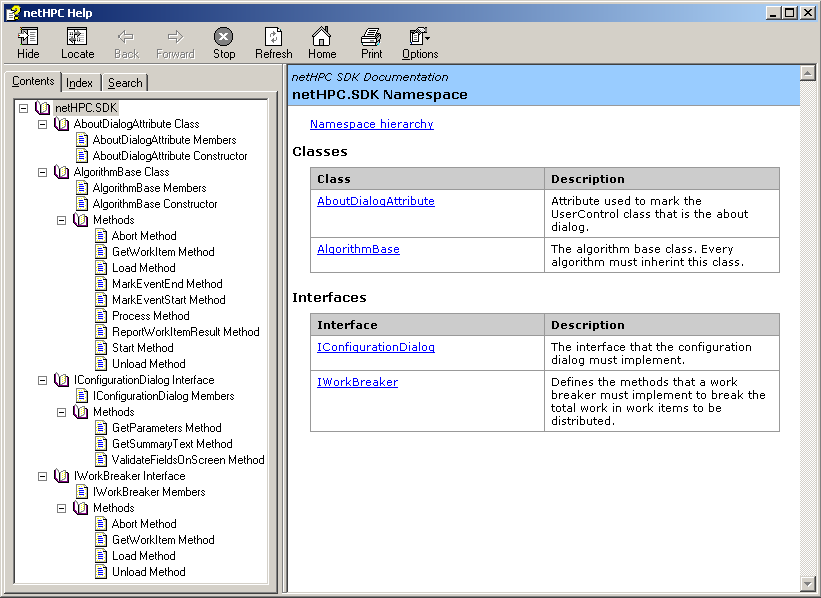


Figura - Arquivo de ajuda do kit de desenvolvimento de software

Fonte: George Luiz Bittencourt (2009)

Este capítulo apresentou o detalhamento de solução desenvolvida no estágio curricular supervisionado. Foi apresentado também o resultado dos testes em campo e as tecnologias da Microsoft que podem ser utilizadas para efetuar uma implantação em larga escala da solução.

CONSIDERAÇÕES FINAIS

A disciplina de estágio curricular supervisionado (ECS), que tem por objetivo desenvolver um sistema utilizando todos os conhecimentos adquiridos durante o curso, foi executada pelo acadêmico na empresa Integral Informática. A Integral atua no ramo de consultoria e desenvolvimento de software no norte catarinense há 13 anos, e como o acadêmico já atua nessa empresa como consultor há 8 anos e o valor da inovação constante é muito forte na companhia, a empresa oportunizou ao acadêmico escolher o tema de seu estágio livremente. O acadêmico escolheu criar um *middleware* de computação em grade em caráter experimental com o objetivo de aprender novas tecnologias, técnicas e também contribuir com a comunidade criando um modelo de objetos em .NET para esse fim.

O objetivo geral do estágio, que era o de construir um *middleware* de computação em grade utilizando Microsoft .NET, foi atingido em sua totalidade. A tela de gerenciamento e o agente que executa o algoritmo foram construídos, o software instalador foi criado e a documentação do kit de desenvolvimento foi elaborada. Além de utilizar o Microsoft .NET, várias outras tecnologias da Microsoft foram também utilizadas para atender os objetivos do estágio. O objetivo de disponibilizar todo o material gerado sob uma licença de código fonte aberto no web site do netHPC também foi atingido.

Dentre os objetivos específicos, o único objetivo que não foi finalizado em sua totalidade dentro do prazo do estágio foi a construção de um segundo algoritmo de exemplo, para ser distribuído junto ao kit de desenvolvimento de software. O acadêmico finalizará esse segundo algoritmo de exemplo e o disponibilizará com os demais artefatos no web site até o final do ano letivo de 2009.

A construção do netHPC iniciou com a revisão da literatura disponível e também com a análise de produtos no mercado. Uma grande dificuldade enfrentada durante essa fase foi a escassez de livros sobre o tema na biblioteca da universidade, porém o acadêmico encontrou vários materiais na Internet que o ajudaram a visualizar o que deveria ser construído. As fases de levantamento de

requisitos, análise de requisitos, projeto e construção foram todas executadas sem grandes dificuldades pelo acadêmico, pois a universidade vem ao longo do curso focando com grande ênfase nelas e com isso a sua execução já está bem fundamentada pelos acadêmicos do curso.

Na fase final do estágio foram realizados testes em campo. Com o resultado destes testes pode-se concluir que a computação distribuída, seja ela no mesmo computador ou entre vários computadores, proporciona uma grande redução na duração do processamento de algoritmos. Esse tipo de processamento, com a atual velocidade no qual a ciência evolui, se mostra cada vez mais necessário para entender fenômenos em tempo hábil.

Entretanto, aqueles que desejam se beneficiar dessa redução não devem mais pensar em execução serial, mas sim em execução paralela e nos problemas que essa traz ao desenvolvimento de software. Problemas como *deadlocks*, falta de sincronização ao acessar objetos e outros, são comuns em aplicativos dessa natureza e muitas vezes são difíceis de serem solucionados.

No estágio atual do desenvolvimento do netHPC, ele ainda não conta com várias funcionalidades, como por exemplo a possibilidade de agendar o início da execução de um algoritmo, possuir mais de um *head node* ativo balanceando a carga entre eles ou ainda, utilizar os processadores presentes nas placas de vídeo ou nas consoles de vídeo game como o Sony® Playstation© 3 ou o Microsoft® Xbox 360©.

Apesar do netHPC ter sido desenvolvido em caráter experimental, é de desejo do acadêmico dar continuidade nesse projeto, visto os benefícios que este pode trazer para a comunidade em geral. Tendo isso em mente, todo o código fonte e outros artefatos serão disponibilizados no web site www.nethpc.com, proporcionando para aqueles que desejarem, participar do desenvolvimento do mesmo.

A execução do estágio foi de grande valia para o acadêmico, tanto do ponto de vista profissional quanto do acadêmico. Do ponto de vista profissional, as várias tecnologias e técnicas de programação aprendidas irão com certeza aumentar o grau de qualidade dos futuros desenvolvimentos do acadêmico. A revisão de literatura também trouxe novos conceitos que serão utilizados pelo acadêmico durante sua vida profissional. Já do ponto de vista acadêmico, com a conclusão do estágio e a elaboração do trabalho, lições foram aprendidas e serão utilizadas pelo acadêmico nas futuras empreitadas planejadas pelo mesmo, como o mestrado e doutorado.

REFERÊNCIAS

BEN-ARI, Mordechai. **Principles of Concurrent and Distributed Programming**, 2 ed. EUA: Addison Wesley, 2006.

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**, 2 ed. Brasil: Editora Campus, 2006.

BRESHEARS, Clay. **The Art of Concurrency**: A Thread Monkey’s Guide to Writing Parallel Applications. EUA: O’Reilly, 2009.

CORMEN, Thomas *et al*. **Introduction to Algorithms**. 2 ed. EUA: The MIT Press, 2001.

CULLER, David, SINGH, J.P., GUPTA, Anoop. **Parallel Computer Architecture:** A Hardware/Software Approach. EUA: Morgan Kaufmann, 1998.

ERIKSSON, Hans-Erik et al. **UML 2 Toolkit**. EUA: Wiley Publishing, 2004.

FOSTER, Ian, KESSELMAN, Carl. **The Grid**: Blueprint for a New Computing Infrastructure, 2 ed. EUA: Morgan Kaufmann, 2004.

GRAMA, Ananth et al. **Introduction to Parallel Computing**, 2 ed. EUA: Addison Wesley, 2003.

GRID Computing. Disponível em http://www.gridcafe.org. Acesso em 16/09/2008.

HOFFER, Jeffrey A. et al. **Modern Database Management**. 8 ed. EUA: Pearson Prentice Hall, 2007.

HUGHES, Cameron, HUGHES, Tracey. **Professional Multicore Programming**: Design and Implementation for C++ Developers. EUA: Wiley Publishing, 2008.

LAW, Averill M., KELTON, W. David. **Simulation Modeling and Analysis**, 3 ed. EUA: McGraw-Hill, 1999.

NETO, João Augusto Máttar. **Metodologia Científica na Era da Informática**. Brasil: Saraiva, 2002.

POWELL, Gavin. **Beginning Database Design**. EUA: Wiley Publishing, 2006.

PRESSMAN, Roger S.. **Engenharia de Software**. Brasil: McGraw-Hill, 2006.

RUMBAUGH, James et al. **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Elsevier, 2006.

SCHAFERSMAN, Steven. **An Introduction to Science**: Scientific Thinking and the Scientific Method. Disponível em: http://www.freeinquiry.com/intro-to-sci.html. Acesso em: 30/07/2009.

SCIENTIFIC method. Disponível em: http://en.wikipedia.org/wiki/Scientific\_method. Acesso em: 30/072009.

SHANLEY, Tom. **The Unabridged Pentium 4**: IA32 Processor Genealogy. EUA: Mindshare: 2004.

SIYAN, Karanjit, PARKER, Tim. **TCP/IP Unleashed**, 3 ed.EUA: SAMS, 2002.

SOMMERVILLE, Ian. **Software Engineering**, 8 ed. EUA: Addison Wesley, 2006.

WOLFS, Frank. **Introduction to the Scientific Method**. Disponível em: http:// teacher.nsrl.rochester.edu/phy\_labs/AppendixE/AppendixE.html. Acesso em: 30/07/2009.

WOOLFSON, M. M, PERT, G. J. **An Introduction to Computer Simulation**. EUA: Oxford University Press, 1999.

APÊNDICE A – RELATÓRIOS DAS EXECUçÕES DE TESTE

