5
Avaliação

No capítulo anterior, apresentamos uma implementação da extensão de fluxo de dados desenvolvida para solucionar o problema enunciado no início deste documento. De posse desse artefato, foi preciso eleger formas de avaliar aspectos como completude das funcionalidades, dificuldade de utilização e características de desempenho. Para avaliar a completude e a dificuldade de utilização, optamos por implementar um aplicação que simulasse os requisitos do executor de fluxo de algoritmos do CSBase. Dessa forma, poderíamos exercitar as novas funcionalidades e analisar sua utilização na prática. Para avaliar o desempenho do protótipo desenvolvido, foram realizadas medidas de desempenho e comparações com outros métodos de transmissão de dados.

Este capítulo apresenta em sua primeira seção um executor de fluxo de dados, comentando detalhes de implementação. A segunda seção apresenta uma análise dos resultados obtidos com as medidas de desempenho. Por fim, a última seção conclui este capítulo com considerações finais sobre as análises realizadas.

5.1
Um Executor de Fluxo de Algoritmos

O executor de fluxo de algoritmos desenvolvido se baseou na infraestrutura de implantação projetada por *Barbosa* [17] e no processo de planejamento original do executor do CSBase, descrito na seção 4.1. O executor é composto por dois *scripts* implementados na linguagem Lua que simulam duas etapas do processo: criação e execução de um plano de implantação. Para avaliar a utilização das novas funcionalidades na prática, foi desenvolvido um conjunto de componentes e planos que exercitavam diferentes características da implementação. Os planos, conforme detalhado no trabalho de *Barbosa*, descrevem passo a passo, etapas de alocação de máquinas, instalação e instanciação de componentes. Dessa forma, foi possível simular o cenário com algoritmos modelados como componentes SCS, e testar a utilização dos novos conectores dinamicamente.

De forma geral, o executor é utilizado em um cenário que apresenta

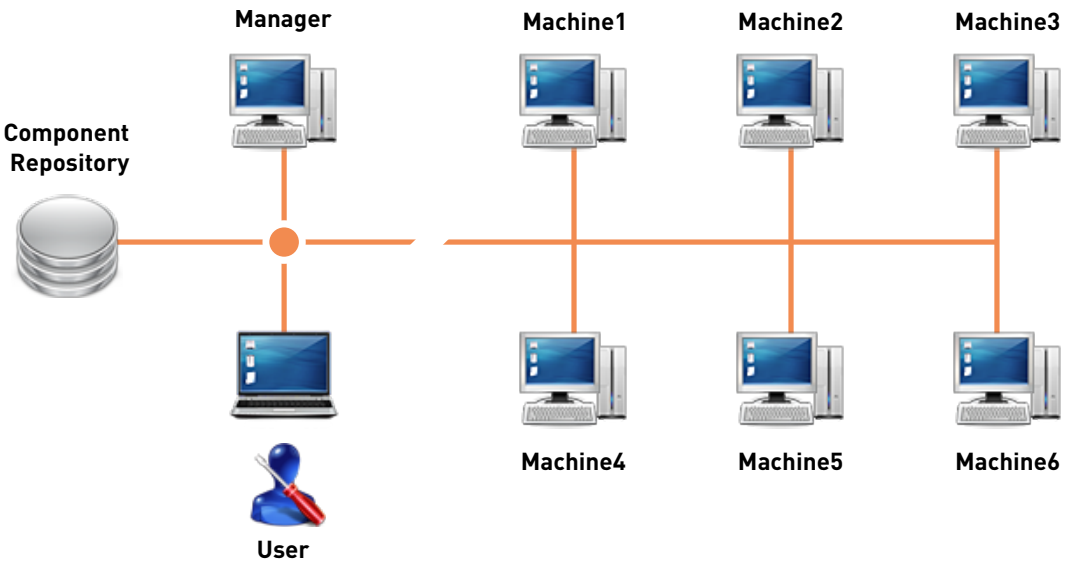

Figura 5.1: Cenário ilustrando as entidades envolvidas em uma execução de fluxo de algoritmos.

um repositório de componentes, um serviço gerenciador de implantação e um conjunto de máquinas disponíveis para utilização, como ilustrado na figura 5.1. O primeiro passo para utilização dessa aplicação consiste em descrever quais componentes serão alocados em quais máquinas, e quais conexões serão realizadas entre eles. A partir dessa descrição o *script createplan* é utilizado para gerar um arquivo que contém o plano de execução baseado na utilização da infraestrutura de implantação. De posse desse plano, o segundo passo consiste em utilizar o *script flowdeploy* para iniciar a execução do plano, resultando na implantação dos componentes e inicialização do fluxo de algoritmos.

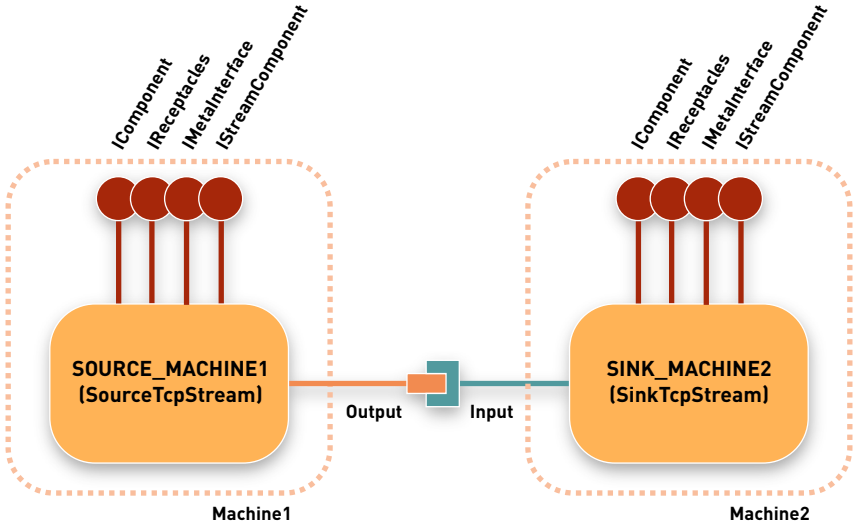

Figura 5.2: Alocação de componentes por máquina.

A figura 5.2 ilustra uma possível alocação de componentes para um fluxo

de transmissão de dados através de conectores TCP. A primeira máquina contém uma instância do componente *SourceTcpStream*, que fornece dados de um arquivo local através do conector *"output"*. A segunda máquina contém uma instância do componente *SinkTcpStream*, que consome dados através do conector *"input"* e os escreve em um arquivo local.

A descrição do fluxo ilustrado na figura 5.2 é composta por duas partes: alocação de componentes em máquinas e conexões entre eles. O código 5.1 descreve a alocação de componentes em máquinas. Na descrição, cada máquina pode conter uma ou mais instâncias de componentes, onde cada instância deve possuir um nome único e um identificador de componente. Com essas informações, o *script createplan* cria um plano com instruções para inicializar um *ExecutionNode* em cada máquina, criar contêineres e carregar componentes descritos.

```
{
  machines = {
    {
      name="machine1",
      components ={
        {
          instance_name = "source_machine1",
          id = {
            name = "SourceTcpStream",
            major_version = 1,
            minor_version = 0,
            patch_version = 0,
            platform_spec = "linux-x86_64"
          }
        }
      }
    },
    {
      name="machine2",
      components ={
        {
          instance_name = "sink_machine2",
          id = {
            name = "SinkTcpStream",
            major_version = 1,
            minor_version = 0,
            patch_version = 0,
            platform_spec = "linux-x86_64"
          }
        }
      }
    }
  },
  ...
}
```

Código 5.1: Descrição das máquinas e seus respectivos componentes.

O código 5.2 ilustra a descrição das conexões, informando os nomes das instâncias de componentes e das portas a serem conectadas. No fluxo representado na figura 5.2, existe apena uma conexão: o conector *"output"* da instância *source_machine1*, com o conector *"input"* da instância *sink_machine2*. Com essas informações o *script createplan* consegue gerar instruções de conexão e inicialização dos fluxos que serão executadas após os componentes terem sido implantados.

```
1 {
2   -- ...
3   connections = {
4     {
5       sink_name="sink_machine2",
6       sink_port="input",
7       source_name="source_machine1",
8       source_port="output"
9     }
10  }
11 }
```

Código 5.2: Descrição das conexões de fluxo de dados entre os componentes

Com a descrição pronta, o processo de geração de plano e implantação do fluxo de algoritmos pode ser realizado por linha de comando, como exemplificado no código 5.3. Como pode ser observado, um arquivo chamado *flow_plan.lua* é gerado e utilizado como entrada do comando seguinte. O arquivo contendo o plano gerado a partir da descrição ilustrada nos códigos 5.1 e 5.2 encontra-se no apêndice A.2.

```
1 $ createplan flow_description.lua > flow_plan.lua
2 $ flowdeploy flow_plan.lua
```

Código 5.3: Criando o plano de implantação e o executando.

O resultado do comando *flow_deploy* é um arquivo *Lua* contendo o nome do plano gerado pela infraestrutura de implantação e as referências de cada componente instanciado. Com essas informações, é possível acessar cada componente diretamente para verificar o estado da execução do fluxo ou até mesmo desfazer a implantação.

```
1 deploy_result = {
2    ["plan_name"] = "Deployer_1.Plan_2";
3    ["components"] = {
4       ["source_machine1"] = "IOR:01ffffff1c00000049444c3a7363732f636f72652
5 f49436f6d706f6e656e743a312e3000010000000000000044000000000102000000000a313
6 2372e302e302e3100e7d5000000163730313235313931323 42f0717200f204b3d030119390
7 0000000001000000000000000800000000 4a414300";
8       ["sink_machine2"] = "IOR:01ffffff1c00000049444c3a7363732f636f72652f4
```

```
9436f6d706f6e656e743a312e3000010000000000000044000000000102000000000a31323
72e302e302e31008fba000000163733303639363831302f0717200f204c330c021510000
00000001000000000000000080000000004a414300";
    };
};
```

Código 5.4: Arquivo com dados da implantação.

5.1.1 Análise do Suporte a Comunicação Através de Fluxo de Dados

O protótipo da extensão apresentada no capítulo anterior implementou novos conectores que possibilitam a comunicação através de fluxo de dados utilizando diferentes mecanismos de transporte. Consideramos a utilização desses conectores, como exemplificado na seção 4.4, simples e adequada para o desenvolvimento de sistemas com esses requisitos, pois esses contam com primitivas bem definidas para produção e consumo de dados. O suporte aos modelos de execução bloqueante e baseado em eventos provê uma flexibilidade interessante para implementações de componentes com diferentes características.

O suporte a diferentes mecanismos de transporte foi implementado através de uma *API* interna para o desenvolvimento de *drivers* de conexão. Essa *API* passou por diversas interações até atingir um mapeamento de métodos que atendesse satisfatóriamente a implementação dos *drivers*. Em nossa concepção, a *API* se mostrou bastante adequada para o desenvolvimento dos *drivers* que suportam *TCP, UDP, named pipes* e *FTC* [37]. No entanto, é possível que futuros *drivers* requisitem certas modificações, como diferentes notificações.

O *driver FTC* foi implementado para avaliar o reuso de bibliotecas existentes com relação a essa *API* interna. A implementação desse *driver* ficou tão simples quanto a implementação do *driver* TCP, pois diversas operações já estavam disponíveis na biblioteca. Inclusive, alguns aspectos da implementação desse *driver*, como a inicialização do laço de eventos, foram posteriormente replicados na implementação dos outros *drivers*.

Atualmente, os *drivers* implementados herdam as características de comunicação impostas pelos mecanismos de transporte. Por exemplo, a política de entrega de dados do *driver* que suporta UDP é a mesma que a definida por esse protocolo. Essa abordagem nos pareceu a mais natural, entretanto um desenvolvedor de *drivers* tem liberdade para fazer implementações com características diferentes.

Uma das limitações da arquitetura proposta é a unidirecionalidade do fluxo de dados. Em outras palavras, os dados sempre fluem do conector *ISource* para o conector *ISink*, independente do mecanismo de transporte utilizado. No

entanto, essa limitação não se mostrou impeditiva para desenvolver o executor de fluxo, pois a bidirecionalidade não é uma funcionalidade utilizada pelos algoritmos atuais.

Uma outra limitação da implementação atual é o suporte a apenas uma fonte de dados nos conectores *ISink*, o que impede o mapeamento de mecanismos de transporte que utilizem múltiplos fluxos de dados em um mesmo conector, como o *GridFTP* desenvolvido por *Allcock et al* [38]. Esse comportamento pode ser simulado com múltiplos conectores, no entanto, não foi investigado.

5.1.2 Análise do Suporte a Implantação Dinâmica

Para suportar a implantação dinâmica de componentes com suporte a conectores de fluxo de dados, o contêiner original da infraestrutura de execução precisou ser alterado. As principais modificações ocorreram na parte de instanciação de componentes, para possibilitar a criação dos novos conectores. Essa experiência deixou claro que uma pesquisa sobre o carregamento de componentes com extensões genéricas precisa ser realizada caso o número de extensões do SCS cresça.

Com a infraestrutura de execução adaptada aos conectores de fluxo de dados foi possível implementar o executor de fluxo de dados apresentado, utilizando descrições de alto nível. Consideramos o sistema implementado adequado e facilmente reutilizável por um editor visual de fluxos como o do CSBase, pois atende todos os requisitos levantados no capítulo 4.

5.1.3 Análise do Suporte a Interrupções e Resumos

Como descrito na seção 4.2, a *API* do conector *ISource* define operações para o suporte a interrupções e resumos. Entretanto, como podem existir algoritmos que não suportem essas operações, a implementação desses métodos é opcional e definida pelo desenvolvedor do algoritmo/componente. Uma limitação derivada dessa política, encontrada na implementação atual, é a falta de mecanismos padronizados para a descoberta desse comportamento em tempo de execução. Esse é um ponto a se considerar em futuras evoluções da extensão proposta, no entanto, não caracterizado como uma limitação impeditiva.

5.1.4 Análise do Suporte a Monitoração de Componentes e Conexões

Como mencionado na seção 4.2, os novos conectores oferecem métodos para introspecção de conexões ativas, o que corresponde parcialmente a funcionalidade de monitoração de execução de um fluxo levantada. Entretanto, mecanismos de monitoração de componentes não fazem parte do escopo desse trabalho. Tais mecanismos podem ser implementados no contêiner de um componente ou como uma faceta do mesmo. Portanto, consideramos que o suporte a introspecção de conexões ativas correspondeu adequadamente aos requisitos levantados, porém, caso um sistema necessite obter informações mais específicas de um componente, como por exemplo, tempo estimado para o término de uma tarefa, o desenvolvimento de uma solução específica se mostra necessário.

5.2 Análise de Desempenho

Esta seção descreve o ambiente de teste e os resultados de dois experimentos realizados para avaliar as características de desempenho da extensão desenvolvida. O primeiro experimento procurou comparar as taxas de transferências obtidas através dos novos conectores, com as taxas obtidas através da transmissão de dados via sequência de octetos *CORBA*. O segundo experimento foi desenvolvido para investigar a utilização do processador durante a transferência de dados nos dois modelos de execução.

5.2.1 O Ambiente de Teste

Os experimentos descritos nesta seção foram conduzidos em uma rede *Ethernet Gigabit*, utilizando duas máquinas com processadores *Intel* conectadas através de um roteador *TP-LINK 300M Wireless N Gigabit Router* (modelo TL-WR1043ND). As duas máquinas executavam o sistema operacional *FEDORA 15* (*Linux kernel* 2.6.42) e utilizavam a versão 1.6.0_31 da *Java Virtual Machine*. A tabela 5.1 apresenta a configuração de *hardware* de cada uma delas.

5.2.2 Taxa de Transferência

O objetivo desse experimento foi investigar se a arquitetura desenvolvida introduzia algum tipo de sobrecarga durante a transmissão de dados. Para isso escrevemos dois componentes SCS, um produtor e um consumidor, que transferiam quantidades de bytes previamente determinadas através dos

Máquina 1	Máquina 2
Processador: Intel Core 2 Duo CPU P8600 2.40GHz **Memória RAM**: 4 GB **Dispositivo de rede**: Realtek RTL8111/8168B PCI-E Gigabit Ethernet	**Processador**: Intel Core i3-2310M CPU 2.10GHz **Memória RAM**: 4 GB **Dispositivo de rede**: Marvell 88E8055 PCI-E Gigabit Ethernet

Tabela 5.1: Configuração das máquinas.

novos conectores, utilizando o *driver* TCP. O componente consumidor era responsável por estabelecer uma conexão e iniciar a transmissão, utilizando os mecanismos descritos na seção 4.2. Os dois componentes recebiam parâmetros com as características específicas de cada teste durante a sua inicialização, como por exemplo, qual modelo de execução utilizar.

Assim, medimos as taxas médias de transferências obtidas com esses componentes e, em seguida, comparamos com as seguintes configurações:

- ***IPERF Benchmark*** [39] - Uma ferramenta de teste que cria fluxos de dados TCP entre dois processos e mede a taxa de transferência entre eles. Essa ferramenta é comumente utilizada como referência para medir desempenho de redes de computadores.
- ***CORBA Octet Sequence*** - Uma ferramenta desenvolvida no âmbito deste trabalho com objetivo de transferir dados entre dois processos através de um ORB utilizando o IIOP/TCP como mecanismo de transporte. Os dados foram representados em IDL como sequências de octetos.

Em todos os casos, o processo produtor foi alocado na máquina 1 e o consumidor na máquina 2. As medidas foram realizadas variando o tamanho do *buffer* nos processos produtores. Em outras palavras, variando o número de *bytes* enviados a cada chamada do sistema. Em cada fluxo de dados o processo produtor enviava 200 megabytes para o processo consumidor.

Os resultados mostrados na figura 5.3 indicam, como esperado, que a implementação da extensão *SCS-STREAMS* não introduziu nenhuma sobrecarga relevante durante a transmissão de dados. Acreditamos que a diferença nas taxas obtidas entre o modelo de execução bloqueante e o modelo baseado em eventos do *SCS-STREAMS* é resultante da quantidade de trocas de contexto das *threads* utilizadas no modelo bloqueante. Quando o tamanho do *buffer* é menor, podemos observar que essa diferença é mais significativa, pois exige mais trocas de contexto para fornecer a mesma quantidade de dados.

No caso da sequência de octetos utilizando IIOP, podemos observar uma sobrecarga significativa na transmissão de dados. Acreditamos que essa

sobrecarga seja causada pela arquitetura de envio de dados e pelo mecanismo de transporte utilizado, provavelmente, resultado da alocação dinâmica de memória, cópia de dados, serialização e deserialização de mensagens executadas pela implementação do ORB. Como no caso anterior, a maior diferença ocorre quanto o tamanho do *buffer* é menor, pois essas operações precisam ser executadas mais vezes para fornecer a mesma quantidade de dados.

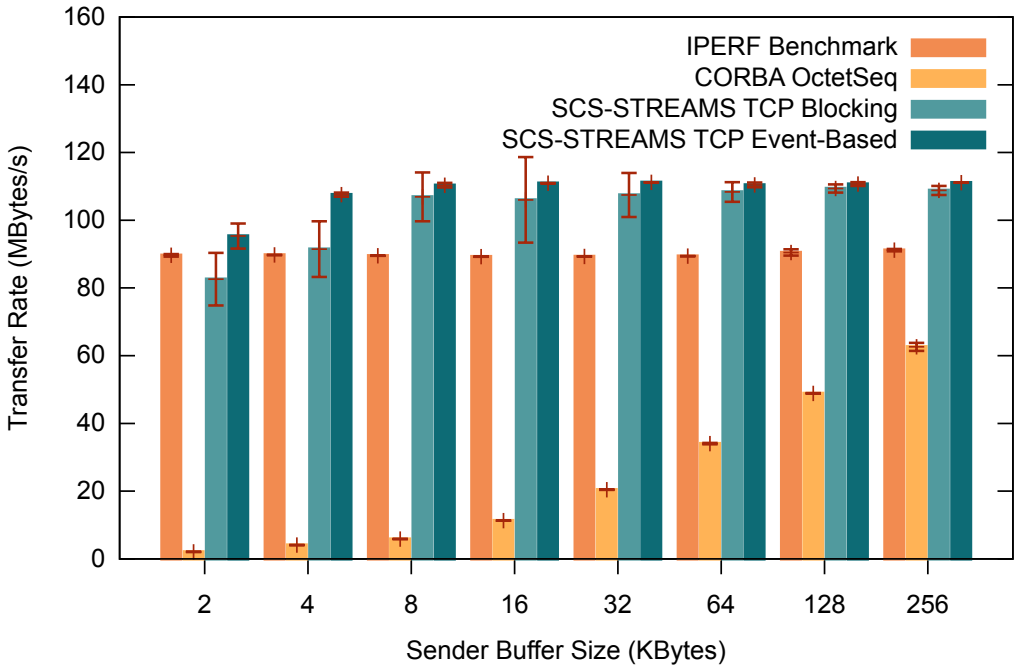

Figura 5.3: Comparação das taxas de transferências obtidas.

5.2.3 Utilização da CPU

O objetivo deste segundo experimento era comparar a utilização do processador nos dois modelos de execução suportados. Para isso utilizamos os mesmos componentes desenvolvidos no teste anterior e medimos a utilização média da CPU no processo produtor durante a transferência de 1 *Gigabyte* nos dois modelos de execução. Neste experimento fixamos o tamanho do *buffer* em 256 *Kilobytes*.

Como pode ser observado no gráfico ilustrado na figura 5.4, o modelo de execução bloqueante utilizou mais recursos do que o modelo baseado em eventos. Esse comportamento era esperado, pois nesse modelo é necessário realizar diversas trocas de contexto para transferir os dados. Além disso, pela mudança no formato da curva perto dos 10 segundos, podemos inferir que o

código executado pela *thread* contendo o laço de produção de dados acabou de enfileirar os pacotes que serão enviados, e que apenas a *thread* de envio ficou ativa. Nessa parte da curva a utilização de recursos é similar a do modelo baseado em eventos.

O resultado obtido foi compatível com o esperado, entretanto, nos parece que o modelo de execução bloqueante necessita um estudo mais aprofundado, visando diminuir a utilização de recursos.

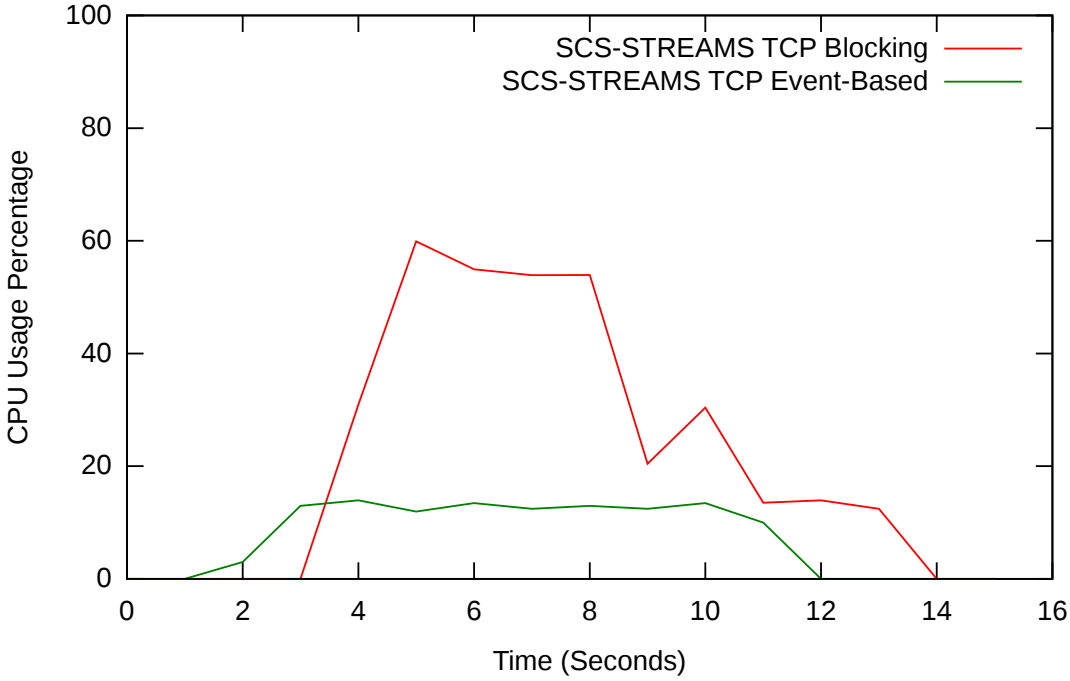

Figura 5.4: Utilização média da CPU do processo produtor nos dois modelos de execução.

5.3 Considerações Finais

Neste capítulo apresentamos uma avaliação da extensão de fluxo de dados proposta para o modelo de componentes SCS. Essa avaliação foi realizada através de análises de diferentes aspectos, como completude das funcionalidades, dificuldades de utilização e medidas de desempenho. A seção 5.1 apresentou um executor de fluxo de algoritmos desenvolvido que foi utilizado para validar as funcionalidades e a utilização da extensão. Nessa seção, algumas limitações foram levantadas, porém, nenhuma delas se mostrou impeditiva, resultando em uma avaliação positiva.

A seção 5.2 levantou características de desempenho do protótipo desenvolvido através de um conjunto de medidas realizadas durante a transferência

de dados. Os resultados obtidos foram comparados aos resultados de outros mecanismos de transmissão, produzindo uma avaliação positiva e validando a arquitetura da extensão para transferências de grande quantidade de dados.

Algumas dificuldades foram encontradas durante a implementação do protótipo e durante a execução das análises, como por exemplo, a dificuldade de identificar e lidar com as falhas do sistema, uma vez que elas ocorriam em diferentes níveis. No protótipo, as falhas podiam ocorrer na extensão desenvolvida, no SCS original, na implementação do ORB CORBA, na máquina virtual de Java ou no sistema operacional. Muitas dessas falhas não são triviais, como uma falha causada pela falta de sincronização entre *threads* que nos deparamos durante testes do protótipo. Em outra ocasião durante os testes, as taxas de transferência obtidas estavam muito baixas e, após uma investigação, descobrimos que a degradação estava ocorrendo devido ao processo de paginação da memória virtual do sistema operacional, pois haviam processos utilizando quantidades razoáveis de memória no computador utilizado.

No entanto, com tempo e perseverança, as falhas foram corrigidas e o resultado final se mostrou positivo diante das análises realizadas.