Desenvolvimento e depuração de aplicações de realidade virtual distribuídas

Development and debugging of distributed virtual reality applications

Marcelo de Paiva Guimarães
Universidade Aberta do Brasil-Unifesp/Programa de Pós-graduação da Faculdade Campo Limpo Paulista
São Paulo, Brail
marcelodepaiva@gmail.com

Valéria Farinazzo Martins
Faculdade de Computação e Informática
Universidade Presbiteriana Mackenzie
São Paulo, Brasil
valfarinazzo@hotmail.com

Diego Colombo Dias
Departamento de Ciência da Computação
Universidade Federal de São Carlos
São Carlos, Brasil
diegocolombo.dias@gmail.com

Luiz Francisco Contri
Departamento de Ciência da Computação
Faculdade Campo Limpo Paulista
Campo Limpo Paulista, Brasil
profluizfiap@gmail.com

Bruno Barberi Gnecco
Corollarium Tecnologia
brunobg@corollarium.com

Abstract— **A busca por alto desempenho e baixo custo tem estimulado o uso de sistemas distribuídos. Tais sistemas são caracterizados por funcionalidades como escalabilidade, tolerância a falhas e capacidade de processamento de alto desempenho. No entanto, o desenvolvimento de aplicativos distribuídos é complexo e requer ferramentas especializadas. Este artigo visa mostrar as técnicas que permitem o desenvolvimento de aplicações de realidade virtual distribuídas baseadas em clusters de computador. Além disso, apresenta o depurador de mensagem GTracer, que visa facilitar o desenvolvimento de aplicações. Essa ferramenta é parte da libGlass, que é uma biblioteca para computação distribuída.**

Palavras-chave: realidade virtual, aglomerados, depurador, sistemas distribuídos e depurador de mensagens

Abstract— **The search for high performance and low cost has stimulated the use of distributed systems. Such environments offer functionality for applications such as scalability, high performance processing capacity and fault tolerance. However, the development of distributed application is complex and requires specialized tools. This paper aims to discuss techniques for the development distributed Virtual Reality applications based on computer clusters. We create the message debugger GTracer to facilitate the development of new applications. This tool is part of libGlass, a library for distributed computing.**

Keywords-word: virtual reality, cluster, debugger, distributed system, message debugger

I. INTRODUÇÃO

As aplicações de Realidade Virtual estão cada vez mais sofisticadas, proporcionando alto grau de imersão e interação para os usuários, e abrangentes, simulando desde conteúdos educacionais, médicos, industriais até esportivos [1, 2, 3]. Tradicionalmente, devido à facilidade de programação, utilizam-se computadores com memória compartilhada para a execução de tais simulações. Esses equipamentos são de alto custo e com escalabilidade limitada (e.g. memória, placas, dispositivos de saída), o que reflete na melhora de desempenho. A tendência atual é a de utilização de sistemas computacionais com memória distribuída (e.g. *clusters*), que são de menor custo, escaláveis e com alta capacidade de processamento. Os sistemas de memória compartilhada se distinguem dos sistemas com memória distribuída principalmente pela forma de distribuição e sincronização dos dados. No caso dos sistemas com memória compartilhada, o *hardware* e a maioria dos *softwares* foram projetados para eles; por isso, a distribuição e sincronização dos dados são realizadas automaticamente pelo sistema operacional, via barramento interno; no caso dos *clusters*, a distribuição e a sincronização dos dados são realizadas por *softwares* de comunicação via rede, sendo a latência maior. Contudo, quando os *clusters* são utilizados com *software* apropriado, eles proporcionam uma alta escalabilidade, podendo ter número ilimitado de nós – mas, isso não garante o ganho de desempenho das aplicações, pois existe a dependência de fatores como latência de comunicação e dependência entre as tarefas.

As vantagens e desvantagens do uso de *clusters* de computadores na área de Realidade Virtual tem sido estudada por vários pesquisadores [4,5,6,7,8,9], o que resultou em diversas soluções, dentre elas os VRClusters [5], que são *clusters* compostos por computadores pessoais que dispõem de *hardware* dedicado a esse tipo de problema, como placas gráficas.

Este artigo discute o processo de desenvolvimento de aplicações de Realidade Virtual baseada em *clusters* de computadores e o depurador GTracer, que é uma ferramenta gráfica desenvolvida neste trabalho para a visualização de mensagens entre os computadores das aplicações construídas com a biblioteca para computação distribuída libGlass. As aplicações tratadas aqui têm como característica proporcionarem um alto grau de imersão e interação para os usuários. Para isso, são projetadas para serem executadas em ambientes de multiprojeção como os CAVEs (CAVE Automatic Virtual Environment) [10] e suportam várias formas de interação, como *trackers*, dispositivos móveis e interfaces naturais [11]. A Figura 1 mostra uma dessas aplicações multiprojetada em um mini-CAVE com três telas. O usuário navega na aplicação utilizando gestos.

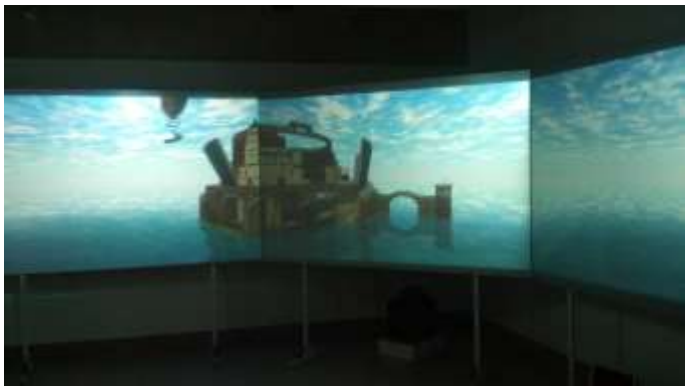

Figura 1. Aplicação executando em um mini-CAVE

As aplicações, como a mostrada na Figura 1, seguem o modelo de execução mestre/escravo com replicação, que é o adotado pela libGlass. Neste tipo de aplicação os dados são replicados em vários nós, o que diminui a necessidade de transmissão de dados. Existem três tipos de nós nesse ambiente: o coordenador, que garante a distribuição e sincronização das informações; os nós de interface, que recebem as interações dos usuários, como, por exemplo, as entradas de uma luva; e os nós de saída, que processam os dados e enviam as imagens para os dispositivos de saída, como um CAVE. Essa divisão de funcionalidade permite a atribuição de tarefas conforme as características do nó. Por exemplo, um dispositivo móvel, que tem baixa capacidade de processamento, pode ser responsável apenas pela entrada de dados. A arquitetura interna da libGlass trata das possíveis falhas, por exemplo, se o nó coordenador deixar de funcionar, automaticamente a tarefa de coordenação é atribuída para outro nó. A Figura 2 mostra esta arquitetura, onde alguns nós tratam o recebimento de interações; o servidor coordena a sincronização e distribuição dos dados; e os outros nós processam os dados e projetam nos dispositivos de saída, tais como projetores e monitores LED/LCD.

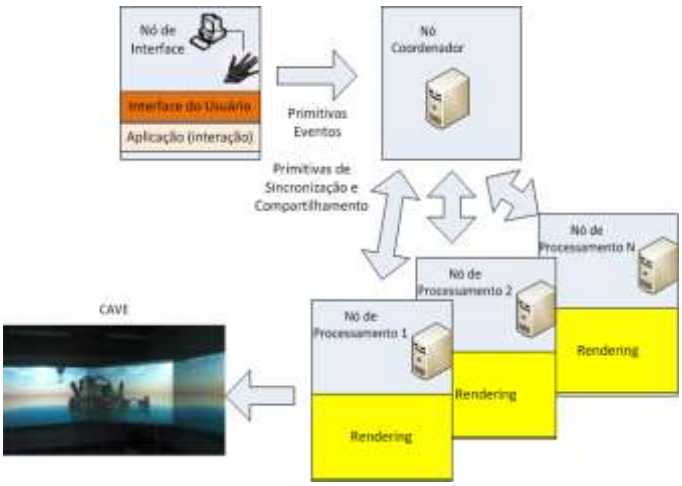

Figura 2. Aplicação executando em um mini-CAVE

O restante do artigo está organizado da seguinte forma: a seção II discute as técnicas para o desenvolvimento de aplicações de Realidade Virtual baseadas em *clusters*; a seção III apresenta o GTracer; por fim, a seção IV mostra as conclusões.

II. DESENVOLVIMENTO DE APLICAÇÕES

As aplicações de Realidade Virtual baseadas em *clusters* podem ser desenvolvidas de maneira automática ou não. Quando construídas de maneira automática, elas são desenvolvidas como se fossem para serem executadas em um sistema computacional com memória compartilhada e uma ferramenta, como um compilador, a transforma para ser executada em um ambiente distribuído. O ambiente Chromium [12] é um exemplo de ferramenta que implementa essa solução. Ele transforma comandos OpenGL de uma aplicação centralizada em fluxos de comandos que são enviados para os nós dos *clusters* gerarem as imagens. Apesar de simples, pois não há necessidade de prever, durante a modelagem e implementação, os problemas referentes a ser um sistema distribuído; a construção de uma ferramenta eficiente de transformação é difícil. Além disso, a aplicação estará limitada aos recursos oferecidos pela ferramenta.

Outra abordagem, que tem gerado resultados melhores do que a anterior, é o uso de comandos que possibilitam a troca de informações e sincronização entre os nós do *cluster*. Essa alteração do código fonte é uma desvantagem, pois deverá ser levada em conta durante a modelagem e implementação. Exemplos de bibliotecas que podem utilizar esta estratégia incluem a libGlass[13], Net Juggler[14] e OpenSG[15].

Nesta abordagem não automática, a estratégia de distribuição e sincronização dos dados pode ser a de Distribuição de estímulos ou a de Cálculo centralizado dos resultados e distribuição [16]. A primeira costuma ser muito simples, consistindo na adição de comandos no código fonte que distribuam os eventos de Entrada (*input*) (e.g. *tracker*, teclado e mouse) para os outros nós. Apesar de simples, pode trazer alguns problemas, já que não existe nenhuma sincronia de dados. Isso significa que, se por algum motivo um dos nós sair de sincronia (perdendo um evento, processando-o

incorretamente ou fora de ordem), o aplicativo não se recuperará. A outra abordagem é mais robusta, e em geral, mais difícil de implementar. Nela, o desenvolvedor trata todos os dados a serem compartilhados ou sincronizados, como a posição e orientação do observador; os estados internos do aplicativo, opções de visualização (e.g *wireframe* ou sólido), entre outros. Os tipos de dados que podem mudar em cada *frame* são os de: controle, que determinam o que desenhar (por exemplo, a direção do ponto de vista do observador em cada nó); e os de mudança do conjunto dos dados (por exemplo, o modelo deve ser atualizado, pois ocorreu uma mudança na textura de um objeto). As alterações desses dados, como, por exemplo, devido à entrada de dados de uma luva, faz com que seja necessária a sincronização para que imagens consistentes sejam geradas.

Independentemente da abordagem de desenvolvimento, o ideal é que ocorra a sincronia dos dados (*datalock*) e a dos *frames* (*swaplock*). A sincronia dos dados deve ser realizada antes do processamento dos dados e a dos *frames* deve ser feita no momento da troca dos *buffers* de imagem. O fato da aplicação ser distribuída, comumente, não exige alteração no tratamento do som. A solução mais simples é usar um nó para a geração do som. Soluções mais avançadas podem criar imersão sonora, usando-se diversas caixas acústicas para gerar som espacialmente localizado.

A abordagem que adiciona a sincronização e distribuição, com o cálculo centralizado dos dados, é a que mais demanda conhecimento de programação, porém, quando bem treinados, os desenvolvedores a implementam com facilidade. O GTracer, que foi criado pelos autores deste trabalho, e é uma ferramenta da libGlass, tem como um dos seus objetivos facilitar o desenvolvimento e aprendizagem no desenvolvimento das aplicações. A libGlass foi construída tendo como meta principal facilitar o desenvolvimento de aplicações distribuídas. Por exemplo, ela permite que qualquer tipo de dados seja compartilhado entre todos os nós de maneira transparente, para isto, basta declarar o dado como sendo do tipo "Shared", ou seja, não será necessário utilizar primitivas de transmissão de mensagens para compartilhar os dados. Para o desenvolvimento de uma aplicação de Realidade Virtual baseada em *clusters* e utilizando a libGlass, é necessário aprender a realizar as seguintes tarefas [16]:

- Inserção das barreiras: que são primitivas que criam pontos de sincronização, nos quais os nós ficam no estado de espera até que todos os nós tenham chegado ao mesmo ponto do programa, para que então, sejam desbloqueados e continuem o processamento. Normalmente, são adicionadas antes da troca dos *buffers* de imagem;
- Funções para o tratamento do ponto de vista: cada nó gera o ponto de vista da imagem de acordo com a sua tela: um nó deve gerar o ponto de vista da direita, outro da esquerda, e assim por diante. Assim, torna-se necessária a criação dessas funções, em que cada uma delas calcula o ponto de visão respectivo. Por exemplo, a função "frente" não modifica a posição ou orientação, enquanto que a função "esquerda" rotaciona o observador em 90º em torno do eixo vertical, e assim sucessivamente;
- Adição das variáveis compartilhadas: para que as imagens sejam coerentes é necessário que estas tenham sido geradas a partir de dados sincronizados. Por exemplo, se a iluminação está habilitada, então todas as imagens das diversas telas devem levar em consideração esse fato. Então, ocorre a necessidade de compartilhar dados como esses;
- Associação do nó à instância: como são utilizados vários nós do *cluster*, então é necessário criar o código que trata do balanceamento das tarefas entre os nós, como, por exemplo, a associação de cada nó a um ponto de vista, o que possibilita a multiprojeção em um CAVE.
- Tratamento das entradas dos dispositivos de interação: os nós que recebem as entradas dos dispositivos devem receber esses dados e enviar para que os nós de processamento gerem as imagens. Podem existir vários nós de entrada, então, por exemplo, um nó pode tratar os dados do teclado, enquanto um outro é responsável pelos dados do *tracker*. Apesar do processamento desses dados ser síncrono, a transmissão é assíncrona.

Essas tarefas são comuns para o desenvolvimento ou porte da maioria das aplicações de Realidade Virtual voltadas para execução em *clusters* e projetadas em dispositivos com os CAVEs, pois, geralmente, possuem geometria estática, requerendo apenas a sincronia dos dados de controle, e da posição e orientação do observador. Por exemplo, a complexidade para transformação da aplicação da Figura 3, que são esferas que circulam ao redor do usuário, e do simulador de avião apresentado na Figura 4, é a mesma.

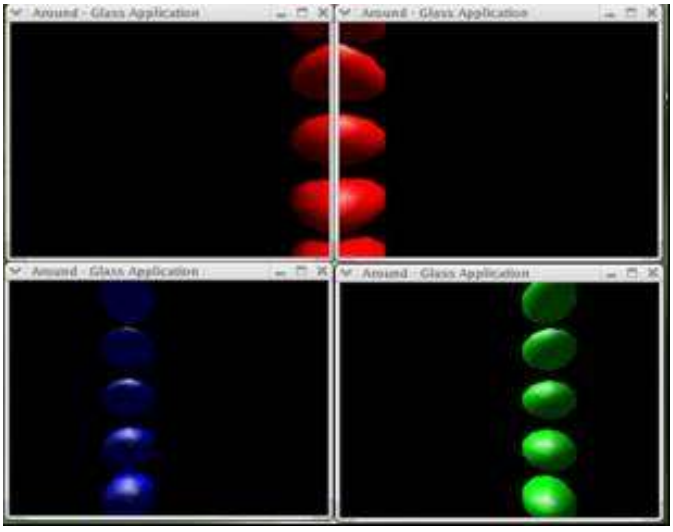

Figura 3. Esferas que circulam ao redor do usuário

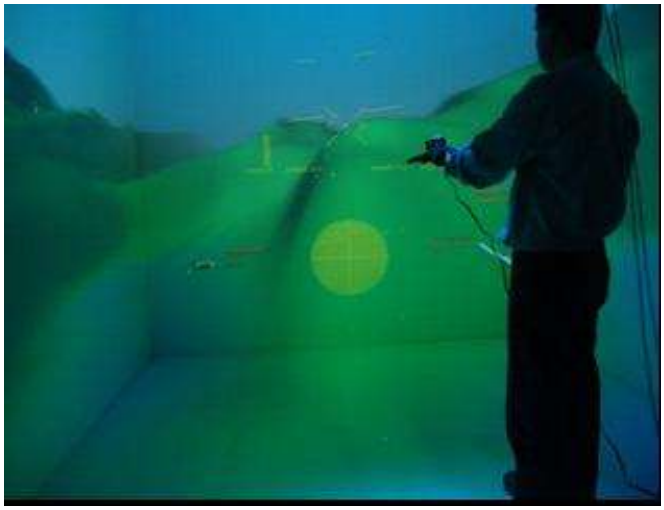

Figura 4. Simulador de aviação GL-117

De forma geral, o código de distribuição e sincronização dos dados da aplicação das esferas e do simulador é semelhante. A Figura 5 mostra os pontos comuns entre essas duas aplicações, o restante do código segue as particularidades de cada uma. Apesar disso, quando a codificação não for realizada de maneira apropriada, a tarefa de depurar o programa sem uma ferramenta apropriada é complexa, pois depende de dados oriundos de diversos nós. Por isso, o GTracer foi desenvolvido. Ele agrega os dados que são transmitidos e sincronizados entre os nós e os representa de maneira visual em uma interface gráfica.

```
…
void esquerda (void)
        {glRotated(90, 0, -1, 0); }
void direita (void)
        { glRotated(90, 0, 1, 0);}
. . .
void glutDisplay (void) {
        // sincronização os dados
        // correção do ponto de vista
        // renderização os objetos
        // barreira para troca dos buffers
        glutSwapBuffers();
}
. . .

int main (int argc, char **argv) {
        // inicialização da libGlass
        // criação das barreiras e das
        // criação das variáveis compartilhadas
        // associação os pontos de vista
        . . .
        glutMainLoop();
}
```

Figura 5. Código para distribuição e sincronização dos dados

III. GTRACER

Como em qualquer tipo de desenvolvimento de *software*, as aplicações baseadas em *clusters* também são propícias a erros. No desenvolvimento de aplicações com memória compartilhada, a identificação de possíveis erros lógicos de programação pode ser facilmente realizada com o auxílio de um depurador.

As linguagens de programação, geralmente, possuem algum tipo de depurador para auxiliar o desenvolvedor na identificação dos erros. No caso de aplicações distribuídas, esse cenário é diferente. A forma como as mensagens são trafegadas entre os nós não pode ser visualizada em um depurador convencional, pois estes se preocupam com a estrutura da linguagem e não com o conteúdo das mensagens.

O desenvolvimento ou porte de aplicações com a libGlass não é dependente de processos distribuídos, e sim, de troca de mensagens. Portanto, um depurador que consiga apresentar informações a respeito das mensagens é de extrema importância.

Existem alguns depuradores com foco no desenvolvimento de aplicações baseadas em *grid* e *clusters*, contudo, a maioria deles visam realizar a verificação de código e processos distribuídos. Hood e Jost [17], em parceria com a NASA, desenvolveram um depurador para ambientes baseados em *grid*. Esse depurador pode encontrar processos distribuídos, mesmo que estes não tenham sido criados sob o controle do depurador, porém, não apresentam as informações relacionadas a troca de mensagens.

Aplicações de Realidade Virtual baseadas em *clusters* possuem grande quantidade de troca de mensagens entre seus nós, pois durante sua execução, qualquer modificação efetuada no ambiente deve ser compartilhada. O sincronismo entre os nós também deve ser realizado a cada *frame* desenhado. Visto que uma aplicação de Realidade Virtual gera em média 90 *frames* por segundo, pode-se afirmar que o sincronismo é uma tarefa de extrema importância. Se houver alguma inconsistência neste tipo de mensagem, a aplicação pode não ser executada da maneira esperada.

Aplicações desenvolvidas com a libGlass podem ser depuradas graficamente por meio do GTracer. O GTracer é um depurador que permite ao desenvolvedor visualizar os dados que trafegam entre os nós de um *cluster*, apresentando informações como data, tipo da mensagem, tipo de dado, origem e destino.

A libGlass permite o armazenamento de todas as mensagens enviadas/recebidas pelos nós em arquivos de *log*, em tempo de execução ou não. Com a opção de depuração habilitada durante a compilação, cada nó cria um arquivo com todos os dados transmitidos e recebidos. Se não houver interesse na depuração, esta deve estar desabilitada, pois ela consome tempo de gravação em disco, o que afeta o desempenho da aplicação, afetando diretamente o grau de imersão e interação. Geralmente, ela é habilitada para a detecção de erros durante o desenvolvimento das aplicações ou para fins de aprendizagem. A Figura 6 apresenta a interface do GTracer, nela o desenvolvedor escolhe o arquivo a ser depurado. O nome do arquivo é formado pela cadeia de

caracter “tcp” seguida do número do processo, e com a extensão “log”, por exemplo, “tcp3204.log” e “tcp3597.log”. Cada arquivo é originário de um nó. Estes arquivos podem ser coletados manualmente dos nós remotos ou pode ser visualizado, durante a execução da aplicação, no nó local.

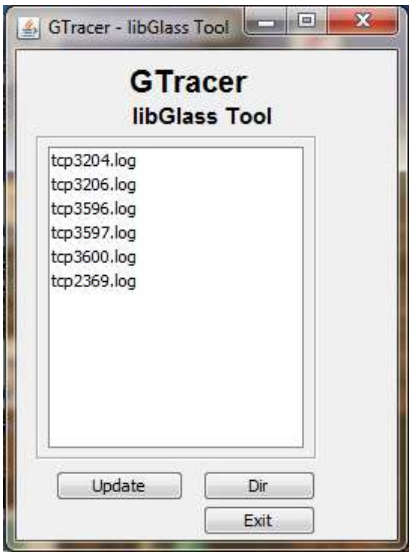

Figura 6. Escolha da aplicação no GTracer

O formato das mensagens gravadas nos *logs* é padronizado, e possuem os seguintes campos:

- Data e horário: momento em que a mensagem é gerada;
- Tipo da mensagem: indica se é uma mensagem recebida ou enviada;
- Tipo de mensagem: informa o recurso utilizado (barreira, compartilhamento ou evento). Uma aplicação pode ter diferentes mensagens do mesmo tipo de dado, por exemplo, a mesma aplicação pode ter várias barreiras;
- Origem/destino: indica o número do nó no qual a mensagem se originou ou para qual foi enviado. A numeração é sequencial de acordo com a ordem de inicialização, no qual o nó número zero (0) é o servidor responsável pela coordenação de todos os outros nós do *cluster*. O nó 0 é responsável pela configuração inicial dos nós de processamento.

A Figura 7 mostra um exemplo do conteúdo de um arquivo de *log* visualizado no modo textual no GTracer. Existem algumas mensagens específicas que não seguem essa padronização, como as que indicam as inicializações internas, o uso de um determinado protocolo de comunicação e as de criação de barreiras. Internamente, implementou-se um *parser* capaz de classificar todas as mensagens.

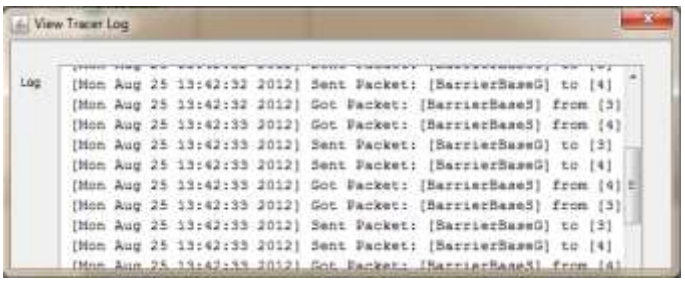

Figura 7. Mensagens do arquivo de log

São geradas várias mensagens de sincronização e compartilhamento para cada atualização de *frame*, então o volume de mensagens torna-se grande em um curto espaço de tempo, o que torna impraticável a análise dos dados no modo texto. Por exemplo, é inviável verificar no modo texto a sequência de comunicação entre os nós a fim de analisar o valor de uma variável compartilhada.

O GTracer, como apresentado na Figura 8, possui a possibilidade de visualização das mensagens em modo gráfico. Esta visualização representa a troca de mensagens entre três nós, no caso o servidor (*Host* 0 - coordenador) e dois outros nós (*Hosts* 3 e 4). O *parser* realizou a leitura do arquivo de *log* em sequência e criou a representação. Cada seta direcional indica o nó de origem da mensagem o respectivo destino. Cada tipo de mensagem é representada por uma cor. Na interface dessa ferramenta é possível, a qualquer momento, inicializar o processo, interromper e inicializar a leitura dos logs. Assim, a aplicação alvo e o GTracer podem ser inicializados, e após um período o sistema de análise de logs pode ser interrompido, o que permite uma análise parcial da mensagem. Além disso, a qualquer momento é possível visualizar as mensagens conforme o filtro de tipo de dado escolhido.

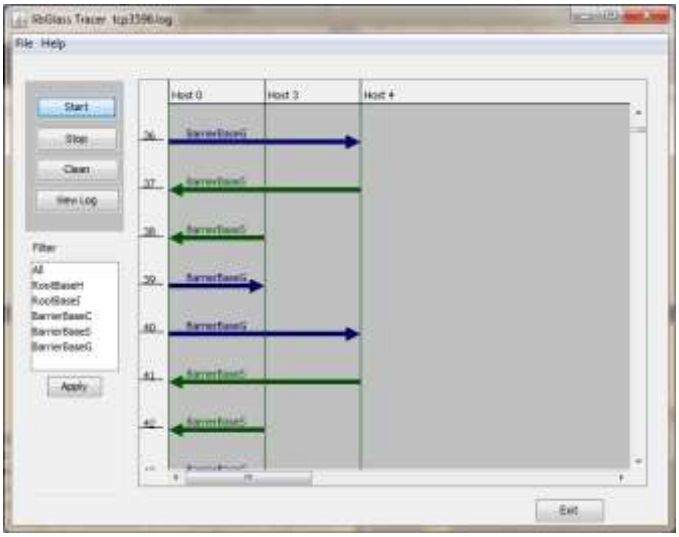

Figura 8. Depuração no GTracer

Quando o *parser* analisa um *log*, automaticamente todos os tipos de dados presentes são classificados e transformados em filtros para visualização, e os nós que receberam/enviaram mensagens são adicionados na interface. A Figura 9 mostra a visualização de um *log* no GTracer com a aplicação de um

filtro. No caso apresentado, as mensagens filtradas referem-se a todos os sinais de sincronização de barreiras recebidos pelo nó servidor (controlador). Então, é possível notar que os nós número 3 e 4 enviaram mensagem. Além disso, está presente também na interface a mensagem no formato textual.

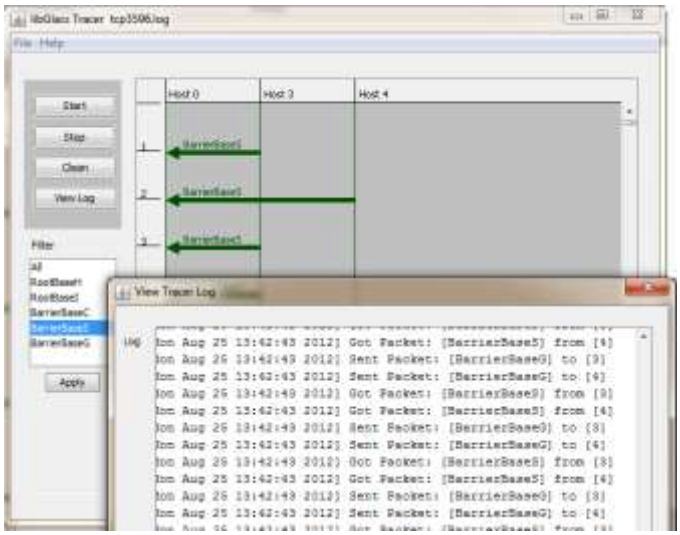

Figura 9. Aplicação de filtros no GTracer

IV. CONCLUSÕES

A libGlass é uma biblioteca para o desenvolvimento de aplicações distribuídas, que permite realizar o desacoplamento de funcionalidades dos nós conforme a característica de cada um (nós de interação, controle e de processamento). Essa biblioteca teve como uma de suas principais metas, a criação de um ambiente de fácil uso, tanto para o desenvolvimento de novas aplicações quanto para as existentes. Isso se tornou um ponto de destaque, pois a maioria das soluções disponíveis requer um grande número de modificações no código fonte, e, às vezes, na arquitetura da aplicação. Além disso, a libGlasss permite a superação de desafios das aplicações com estereoscopia, que é a sincronização de dados (*datalock*) e a de conclusão de *frames* (*framelock*). O desempenho das aplicações já desenvolvidas com a libGlass teve como limite a capacidade de processamento da placa gráfica utilizada.

Como a abordagem de desenvolvimento adotado pela libGlass não é automático, pois proporciona melhor resultados do que a automática, visto que não limita os recursos a serem utilizados durante o desenvolvimento, então torna-se necessária ferramentas como o GTracer. Esse depurador de mensagens auxilia o desenvolvimento de novas aplicações e facilita o treinamento de desenvolvedores. Depuração de mensagens não é trivial. O Gtracer apresenta uma solução diferente das soluções convencionais, que, normalmente, focam na depuração de processos distribuídos, e não na troca de mensagens entre os nós.

Os próximos passos deste trabalho é aprimorar o GTracer, adicionando funcionalidades como a coleta automática dos *logs* dos nós remotos e apontamento automático de falhas (e.g. *deadlock*). Além disso, como está sendo adicionado na libGlass o suporte para computação em nuvem, então o GTracer também poderá ser utilizado nesse contexto.

REFERÊNCIAS

[1] Q. Y. Kai-Hu, L. Xin-Jian.”Application of Computer Virtual Reality Technology in Modern Sports”. Third International Confererence in Intelligent System Design and Engineering Applications (ISDEA), 16-18 Jan, pp. 362- 364, 2013.

[2] A.G.Abulrub."Virtual reality in engineering education: The future of creative learning",Global Engineering Education Conference (EDUCON), 4-6 April, pp.751-757, 2011.

[3] A.M. C. Souza."Handcopter Game: A Video-Tracking Based Serious Game for the Treatment of Patients Suffering from Body Paralysis Caused by a Stroke".Virtual and Augmented Reality (SVR), 28-31 May 2012, pp. 201-209, 2012.

[4] P. Lin, Z. Pan, J. Yang, J. Shil, “ Implementation of a Low-Cost CAVE system Based on Networked PC”, Virtual Environment on a PC Cluster Workshop 2002, Protvino, Russia, September, pp.33-40, 2002.

[5] J. A. Zuffo, L. P. Soares, M. K. Zuffo, R D. Lopes, “ CAVERNA Digital – Sistema de Multiprojecão Estereoscópico Baseado em Aglomerados de PC’s para Aplicações Imersivas em Realidade Virtual”, In Procedings of the 4th Brazilian Symposium on Virtual Reality - SVR’01, Florianopolis, SC, Brazil, Oct. 16-19, pp. 139–147, 2001.

[6] M. Zuffo, L. P. Soares, P. Bressan, M. P. Guimarães, “Commodity Cluster for Immersive Projection Environments”, Tutorial.SRV – Symposium on Virtual Reality, Fortaleza, 2002.

[7] SGI Graphics Cluster TM: The Cluster Architecture Challenges, the SGITM Solution, Visualization Solutions Development Group, White Paper, 2001.

[8] B. Schaeffer, C. Goudeseune, “Syzygy: Native PC Cluster VR”, IEEE Virtual Reality Conference, Aug, pp.25-22, 2003. Disponível em: http://www.isl.uiuc.edu/syzygy.htm

[9] J. T. Klosowski, “Chromium: A Stream Processing Framework for Interactive Graphics on Clusters”, In Proceedings of ACM SIGGRAPH 2002, pp. 693—702, 2002.

[10] C.CRUZ-NEIRA, D. J. SANDIN, T. DEFANTI, R. KENYON,J. C. HART. "The CAVE: Audio Visual Experience Automatic Virtual Environment", Communications of the ACM, vol. 35(6), pp. 64–72, 1992.

[11] R. HarperT.Rodden, Y. Rogers, A.Sellen(eds). “Being Human-computer interaction in the year 2020”. Isbn: 978-0-9554761-1-2, Publisher: Microsoft Research Ltd, 2008.

[12] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, P. Hanrahan, “WireGL: a scalable graphics system for clusters”, In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01), ACM, New York, NY, USA, pp.129-140, 2001.

[13] M. P. Guimarães, B. B. Gnecco, M. Cabral, L. Soares, M. Zuffo, “Synchronization and data sharing libray for PC clusters”, Workshop on Commodity Clusters for Virtual Reality - VR-CLuster'03. Los Angeles, USA, Marc. 22th-26th, 2003.

[14] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin, “Net Juggler: Running VR Juggler with Multiple Displays on a Commodity Component Cluster”, In Proceedings of the IEEE Virtual Reality Conference 2002 (VR '02), IEEE Computer Society, Washington, DC, USA, 273-274, 2002.

[15] OpenSG, “OpenSG Welcome”, 2013, Disponível: http://www.opensg.org/.

[16] M.P.Guimarães."Um Ambiente para o Desenvolvimento de Aplicações de Realidade Virtual baseadas em Aglomerados Gráficos". Tese de doutorado. Escola Politécnica da Universidade de São Paulo, 2004.

[17] R. Hood, G. Jost, "A debugger for computational grid applications," Heterogeneous Computing Workshop, 2000, (HCW 2000) Proceedings, 9th , vol., no., pp.262-270, 2000.