PCS-5732 - Cliente-Servidor arquitetura de modelos de programação

Profa. Graça Bressan

Logistical Network Stack
Uma infraestrutura para Data Grid

Claudio Penasio Junior
Jefferson Máximo da Silva
Sílvio Luís Marangon

1º período 2004

Sumário

Figuras

Tabelas

1 Objetivo

Por décadas, os modelos básicos utilizados para o compartilhamento de recursos foram o Computer Center Model e o Internet Model. As diferenças entre estes dois modelos tende a gerar divergências entre possíveis soluções, o que leva a necessidade de procurarmos por novas soluções para o desenvolvimento de infra-estrutura para o compartilhamento de recursos para a comunidade científica. Nos últimos anos um novo paradigma para compartilhamento de recursos de informática, o Grid, vem tornando-se mais comum [5].

Neste contexto, esta monografia apresentará a infra-estrutura de software proposta pelo Logistical Networking Project (LoCI Laboratory, University of Tennessee), a qual pretende dar suporte para o armazenamento e para a transmissão de dados para Data Grid.

Nos introduziremos o Logistical Network Stack, um conjunto de sistemas e protocolos que permitem a um aplicação de Grid, ou de rede, acessar recursos de armazenamento com alta-performance e independência de localização[2].

2 Introdução

O termo *Grid* foi concebido na década de 90 como uma infraestrutura de computação distribuída para computação cientifica e de engenheira. Atualmente a computação em *Grid* é tratada como uma nova geração de infraestrutura para tecnologia de informação, que irá transformar a forma de como é realizada a computação, comunicação e colaboração em sistemas de informação. Espera-se que muitos *Grids* irão surgir no futuro, tendo cada um com seu próprio propósito, sendo compartilhados por instituições ou comunidades virtuais com um mesmos interesses [19]. Para melhor definir o que é um Grid vamos apresentar outras definições encontradas na literatura:

"*Grid* é uma infraestrutura emergente, que faz o processamento e o acesso a dados serem possíveis de qualquer lugar e a qualquer horário, sem que alguém seja obrigatoriamente notificado". [20]

Computação em *Grid* é uma coleção de recursos heterogêneos e distribuídos possibilitando que sejam utilizados em grupo para executar aplicações de larga escala. [21]

A computação em *Grid* pode prover vários benefícios, como:

- Agregar recursos de computação de toda a infraestrutura de informação da instituição, não importando sua localização.
- Melhorar a qualidade dos serviços de rede.
- Aumentar velocidade de resposta dos serviços disponíveis.
- Permitir o acesso e compartilhamento de bases de dados remotamente.
- Maior robustez do sistema de informação perante falhas ou desastres.

Enquanto abordagens tradicionais focam em aperfeiçoar o uso de recursos computacionais já escassos, acontece um crescimento em todas as áreas. Os recursos computacionais tiveram um significativo aumento em: poder de processamento, banda de comunicação e armazenamento. Diante deste cenário pesquisadores apresentam um problema diferente: Como criar novas arquiteturas que possam aproveitar os recursos computacionais já existentes?. O nosso trabalho aborda os aspectos da arquitetura chamada de *Logistical Networking*, que de um modo geral explora a utilização dos recursos computacionais existentes para criar uma infra estrutura e poder compartilhar estes recursos de computação.

Os recursos compartilhados são a essência de um sistema de redes e o objetivo do sistema de redes de computadores é tipicamente baseada em comunicação, com a ocorrência de transmissão de dados entre os sistemas, onde o recurso de *bandwith* é exigido para implementar o núcleo da transmissão. Com um simples sistema de conexão ponto a ponto

entre *end-system* é possível construir redes, e prover serviços como exemplo o sistema de *e-mail*, que são gerenciados completamente pelos *hosts* e são usados pela comunidade. Normalmente, os computadores conectados a rede são definidos em termos de comunicação como *end-systems*.

Em redes públicas, como a Internet que compartilham a mesma infra-estrutura que esta potencialmente aberta a qualquer usuário. Mas esta forma de compartilhar, só é possível através da redes IP, os roteadores e *switchs* provêem a *bandwith* necessária e representam os recursos computacionais mais significantes a soma destes elementos computacionais intermediários permitem administrar a rede controlar e alocar *bandwith* em *links* individuais, tornando o recurso compartilhado. Aplicações *end-system* muito simples que requerem conexão podem fazer uso da complexa *Internet Global* utilizando de *bandwith* e infraestrutura de administração.

3 Conceitos de Logistical Networking

Logistical Networking pode ser definido como otimizador e *scheduling* global de armazenamento, transferência e computação de dados em um modelo que considere todos os recursos da rede, em contraste com um modelo mais tradicional, o qual não explicitamente modela os recursos de armazenamento e computação na rede [1].

O projeto *Logistical Networking* é chamado assim por uma analogia aos sistemas de armazenagem, depósitos e distribuição normalmente utilizadas nas logísticas do exército e de atividade industrial.

Active Networking é uma abordagem para avançados sistemas de redes, que dá um grande passo nesta direção permitindo aos usuários e aplicações compartilharem do poder computacional dos roteadores e *switchs*. O *Active Network* cria o tipo de capacidade de administração necessária para fazer uso da explosão crescente dos recursos de computação e armazenamento, que estão disponíveis para serem usados como parte da infra-estrutura de rede.

A característica principal do *Logistical Networking*, aborda um paradigma um pouco diferente do *Active Networking* - é o modo agressivo que recursos são expostos para aplicação usar. Isto é exemplificado em nossa pesquisa atual, que foca o uso de armazenamento de redes e usos de protocolos simples chamado *IBP (Internet Backplane Protocolo)* [4]. O *IBP* implementa serviço de armazenamento que aplicações podem usar para propósitos de *Logistical Networking*, enquanto permite as aplicações de redes a fazerem o uso de recursos alocado a elementos intermediários, mas não assume que os clientes destes serviços sejam localizados por elementos intermediários como por exemplo roteadores. Ao contrario das maiorias das abordagens *Active Networking*, o *IBP* controla recursos que podem ser usados por qualquer elemento de *Active Networking* ou diretamente através de *end-systems*. *Logistical Networking* surge para habilitar uma grande variedade de diferentes modos de usar armazenamento na rede.

4 Logistical Network Stack

O *Network Storage Stack* ou Pilha de Armazenamento em Rede, é um conceito que possibilita a implementação do *Logistical Networking*, que é uma nova visão de armazenamento em rede. O *Logistical Networking* possui uma abordagem incomum de que o armazenamento pode ser usado para aumentar a transmissão de dados como parte de um recurso unificado de uma estrutura de rede.
O *Network Storage Stack* tem como principal foco criar uma camada de abstração para que o recurso de armazenamento em rede faça parte da WAN (*wide-area-network*) de forma eficiente, flexível compartilhada e escalável.

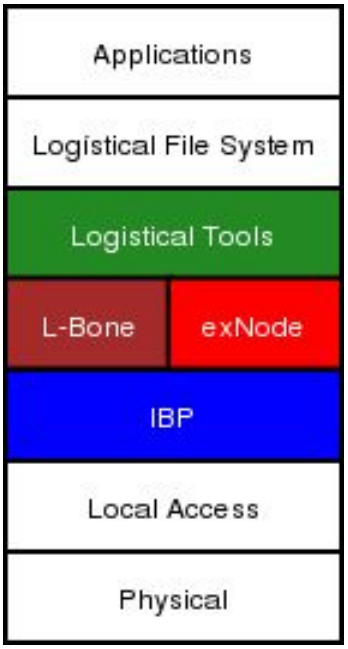

Figura 4-4.1 ***The Network Storage Stack***

Dois aspectos importantes desta abordagem em camadas é que cada camada abstrai as camadas inferiores, ao mesmo tempo que expõe seus recursos às camadas superiores[13].
Neste Trabalho trataremos as três camadas centrais do *Network Storage Stack*, uma vez que as duas camada de baixo são *Hardware* e Sistema Operacional, e as duas de cima ainda estão em desenvolvimento.

4.1 IBP

O *IBP (Internet Backplane Protocol)* é a camada mais baixa da *Network Storage Stack*, e foi projetado para ser a mínima abstração do serviço de armazenamento necessário pelo *Logistical Networking*. O *IBP* é um *software* de servidor (*IBP server*) do tipo *daemon* e uma biblioteca cliente que permite que os donos do conteúdo a serem armazenados possam colocá-los na rede. O *IBP* Pode ser visto como um mecanismo para gerenciar *buffer* de comunicação e "arquivos" remotos.

4.1.1 IBP API

O *IBP API* foi projetado para ser a mínima abstração de armazenamento para servir as necessidades do *Logistical Networking*, cujas operações fundamentais são:

- Alocar *um byte-array* para o armazenamento do dado
- Mover dados de um remetente para o *byte-array*
- Entregar dados do *byte-array* para o receptor.

O *IBP Client API* possui sete chamadas de procedimentos, divididas em três grupos

Storage Management	Data Transfer	Depot Management
IBP_allocate IBP_manage	IBP_store IBP_load IBP_copy IBP_mcopy	IBP_status

Tabela 4-4.1 Camadas de procedimentos *IBP Client*

Alocação

IBP_allocate: É utilizado para requerer uma alocação (uma quantidade de espaço em uma máquina por um espaço de tempo). Se o deposito (*IBPserver*) tem espaço e o tempo requisitado menor ou igual a duração máxima determinada na política do depósito, então a requisição é atendida e o depósito envia um "*capabilities set*" (chaves) para o usuário. Estas chaves garantem escrita (*store*), leitura (*read*), e acesso de gerenciamento do espaço.

```
IBP_cap_set  IBP_allocate(IBP_depot  depot,  IBP_timer  timeout,
ulong_t size, IBP_attributes attr)
typedef struct ibp_depot {
     char    host[IBP_MAX_HOSTNAME_LEN];
     int      port;
} *IBP_depot;
```

```
typedef struct ibp_attributes {
time_t      duration;    /* lifetime of the capability */
int       reliability;    /* reliability type of the capability
*/
int             type;    /* capability storage type */
} *IBP_attributes;
```

Atributos de alocação:

Duração : Permanente ou Período pré-determinado.

Reliability: IBP_STABLE ou IBP_VOLATILE

Determina se o servidor pode remover os dados a qualquer momento (volátil) ou mantê-lo pelo tempo de vida do buffer (estável).

Tipos de acesso :

IBP_BYTEARRAY : adiciona dados somente no fim da array.
IBP_FIFO.
IBP_CIRQ : Fila circular.
IBP_BUFFER : adiciona dados no inicio do buffer e acesso restrito.

Retorna capabilities, que identificam o dado armazenado.

```
/* Definition of the capability */
typedef char* IBP_cap;

/* Definition of the capability set */
typedef struct ibp_set_of_caps {
     IBP_cap readCap;        /* read capability */
     IBP_cap writeCap;       /* write capability */
     IBP_cap manageCap;      /* manage capability */
} *IBP_set_of_caps;
```

ibp://hostname:port/key/WRMKey/WRM

host	nome do host.
port	nº da porta.
key	Valor que pode ser considerado como nome do buffer.
WRMKey	WriteKey/ ReadKey/ ManageKey - valor que permite acesso aos dados.
WRM	Indica se WRMKey é para : WRITE, READ ou MANAGE

Exemplo:

```
ibp://planetlab2.cs.wisc.edu:6714/IBP-55031061718223314901067733618113685917786951805196800564220244974891860674424159454312127963843541592393205006813920438913676665230879762786355303450003/655263292/READ
```

Leitura e Escrita:

IBP_store: Disponibiliza recursos para o usuário realizar escrita no dado na alocação.

IBP_load: Uma vez que o dado foi escrito na alocação o usuário pode utilizar o *IBP_load* para ler o dado a partir da alocação.

IPB_copy: Permite que o usuário realize uma transferência externa que é enviar os dados de uma alocação para outra (ponto-a-ponto) sem trazer o dado de volta para o cliente. o *IBP_copy* usa *TCP*.

IBP_mcopy: Permite que o usuário realize uma transferência externa ponto-a-ponto ou ponto-a-multiponto. O *IBP_mcopy* pode utilizar diversos protocolos, inclusive *TCP*, *UDP* (confiável ou não), e protocolos *non-IP*. Os depósitos permitem módulos de *plugins* para protocolos adicionais.

```
Leitura/Escrita entre IBP buffer e memória.
     ulong_t  IBP_store(IBP_cap  write_cap,  IBP_timer  timeout,  char *data, int size)
     ulong_t IBP_load(IBP_cap read_cap, IBP_timer timeout, char *buf, int size, int offset)

Leitura/Escrita entre dois ou mais IBP buffers.
     unsigned long int IBP_copy(
       IBP_cap source, IBP_cap target, IBP_timer timeout,
       unsigned long int size, unsigned long int offset);
     ulong_t IBP_mcopy(
       IBP_cap sourceCap, IBP_cap targetCap[],
       unsigned int capCnt,
       IBP_timer src_timeout, IBP_timer tgt_timeout,
       ulong_t size, ulong_t offset,
       int type[], int port[], int service);
```

Tipos de mcopy		
Source	**Targets**	**Description**
TCP DM_UNI	DM_SMULTI	Sequencial mcopy
	DM_MULTI	Round-robin fashion mcopy
	DM_PMULTI	Threaded mcopy
UDP DM_BLAST	DM_MBLAST	Thread UDP Based mcopy

Tabela 4-4.2 Tipos de mcopy

Gerenciamento

IBP_manage: Permite ao usuário alterar as propriedades da alocação, como por exemplo: requisitar o prazo de expiração, aumentar ou diminuir o tempo de validade dos dados e também podemos aumentar ou diminuir o espaço alocado em buffers.

```
int IBP_manage(IBP_cap manage_cap, IBP_timer timeout, int cmd,
int capType, IBP_status info);
Valores para cmd :
     IBP_INCR     Incr. o contador de referências da Capability.
     IBP_DECR     Dec. o contador de referências da Capability.
     IBP_CHNG     Altera : maxSize e duration.
     IBP_PROBE    Verifica a propriedade.
Valores para capType :
     IBP_READCAP
     IBP_WRITECAP
```

4.1.2 IBPserver

O *IBP Server*, em vários artigos também tratado como *IBP Depot* tem como unidade de armazenamento o *byte-array e o time-limited;* permite que os dados sejam armazenados em uma localidade, adicionando o controle temporal a transferência de dados, além do espacial. Neste conceito não existe a idéia de arquivos ou diretórios. A alocação do tipo *byte-array*, é como se fosse uma chamada de função *malloc()* de rede, onde o cliente faz a requisição de alocação a um *IBP Storage Server* específico, e se obtiver sucesso, é retornado três *strings* de texto criptografadas (conhecidas como *capabilities*) para leitura, escrita e gerenciamento. As *capabilities* podem ser usadas por qualquer cliente na rede, e podem ser passadas livremente de um cliente para outro, ou seja eles podem trocar *capabilities* livremente sem registrá-las no servidor.

Como estratégia de alocação de dados *o IBP server* possui um *design* que permite que os servidores recusem uma requisição de alocação, dependendo do tempo ou espaço requeridos. Na política dos depósitos qualquer alocação deve ser menor que a curva definida por “espaço*tempo = K” onde K varia linearmente com o espaço livre remanescente.[14].

Características do *IBP*:

- Transmite e armazenas *arrays* de *bytes*.
- Gerencia o armazenamento remoto.
- Pode ser utilizado por *Active Networking Elements* e *end-systems*.

4.2 LBone e exNode

O *LBone (Logistical Backbone)* é uma aplicação de camada que permite a descoberta de recursos, enquanto o *exNode* é uma estrutura de dados, como se fosse um *inode* do *Unix* em rede.

4.2.1 LBone

O grupo de desenvolvimento do *LBone* do Laboratório *LoCI* da Universidade do *Tennessee*, diversas instituições em vários países do mundo estão criando novos servidores *IBP* e registrando no serviço *LBone*.

Atualmente o *LBone* é baseado no sistema de diretórios *LDAP*, onde as informações como proximidade de rede para qualquer ponto de acesso na Internet pode ser calculado em tempo real. [15]
O *Logistical Backbone* (*LBone*) mantém um diretório do *IBPserver* além de metadados sobre estes servidores. Ele tem um arquivo para cada depósito. Um dado importante é que o *LBone* armazena os dados padrões do *IBP* incluindo ***hostname***, ***porta***, quantidade de ***soft*** e ***hard storage*** e o período máximo de alocação para cada depósito.

O *LBone* permite que clientes busquem os depósitos de dados, bem como os dados nos depósitos. Os clientes também podem realizar consultas com o *LBone* a respeito de características, como capacidade mínima de armazenamento, requisitos de duração e requerimentos de proximidade. Outra característica do *LBone* é permitir que os depósitos de dados *IBP* se registrem a si mesmos no *LBone*.

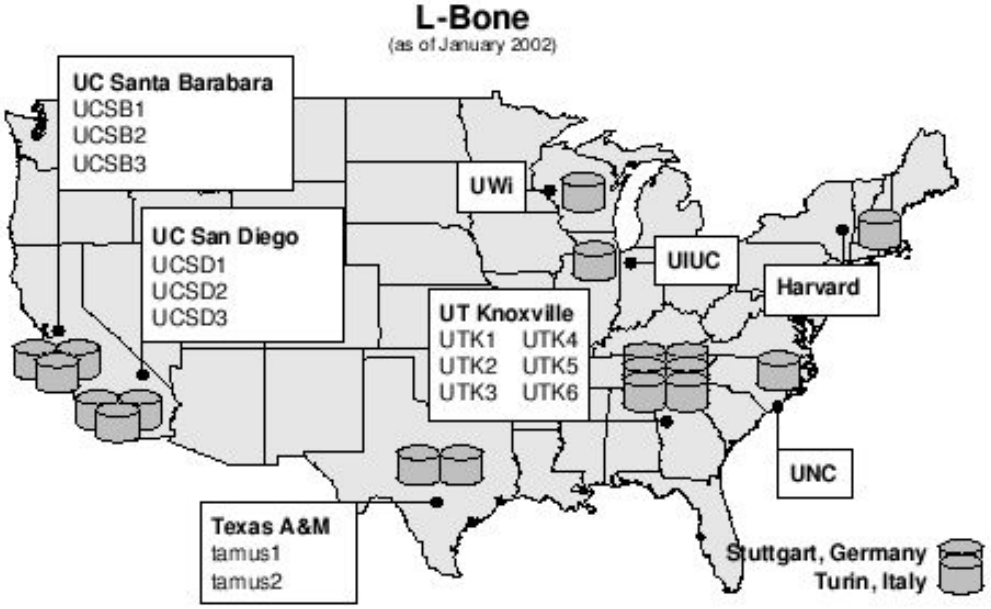

Figura 4-4.2 LBone

	Total Storage (GBs)	Hard Storage (GBs)	Soft Storage (GBs)
Listed	**34874.26**	**9345.58**	**34874.26**
Offline	**10617.77**	**1405.78**	**10617.77**
Online	**24256.49**	**7939.80**	**24256.49**
Used	**733.89**	**650.11**	**83.78**
Allocable	**23522.57**	**7147.75**	**N/A**
Percentage of Online, Allocatable Storage Available	**96.96 %**	**91.08 %**	**N/A**

Tabela 4-4.3 dados coletados 21/05/2004 17:04 Zona Horária: América – Brasília

4.2.2 LBone API

O *LBone* reutiliza o *ibp_depot struct(char host[256]* e *int port)* e define o depósito equivalente para o ponteiro *ibp_depot*.

A estrutura de dados primária do usuário *LBone* é a *LBONE_request*:

```
type struct {
     int numDepots;
     unsigned long stableSize;
     unsigned long volatileSize;
     double duration;
     char *location;
} LBONE_request;
```

A estrutura *L-BONE_request* possui a informação passada para o servidor de acordo com o critério de busca do cliente.

numDepots é o número máximo de depósito a retornar. Ele pode retornar um número menor de depósitos do que o total. Se você quiser receber todos, deve especificar *ALL_DEPOTS*.

stableSize e **volatileSize** indica a quantidade mínima em *megabytes* de cada necessidade para incluir no *array* de retorno. Se ambos são diferentes de zero, então a busca é do tipo *AND*. Se um dos dois é zero, ele busca para o não zero .

duration é o número mínimo de dias que o usuário precisa do espaço. Este valor pode ser maior ou igual a zero. É do tipo *double* para o usuário poder especificar quantidades parciais de dias.

location permite ao usuário digitar um par de argumentos que são uma palavra chave e um valor para determinar onde ele quer armazenar os dados e o critério mínimo do ambiente de armazenamento.

4.3 ExNode

O *exNode* é uma estrutura de dados análoga a um *inode* no *Unix*. Enquanto no *Unix* o *inode* agrega blocos de um único volume de disco para compor um arquivo (figura 4-3).

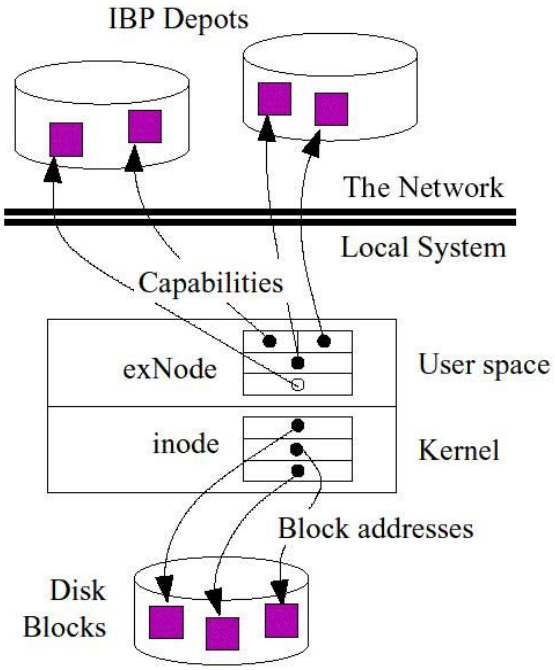

Figura 4-4.3 exNode

O *exNode* agrega o *IBP byte arrays* para compor um entidade lógica, que pode ser utilizada como um arquivo. O *IBP buffer*, pode ter qualquer tamanho e pode ser replicado, Figura 4-4 mostra diferentes formas de armazenamento de um arquivo de 1000 *bytes*. No primeiro exemplo, ele insere todos os 1000 *bytes* no *IBP server*, no segundo exemplo, ele tem duas réplicas do arquivo e coloca uma no depósito B e outra no C, e finalmente no terceiro exemplo, ele tem duas réplicas do arquivo, sendo que na primeira

réplica ele a divide em dois segmentos e os coloca respectivamente no depósito A e D, e a segunda réplica é dividida em 3 segmentos colocados respectivamente nos depósitos B, C e D.

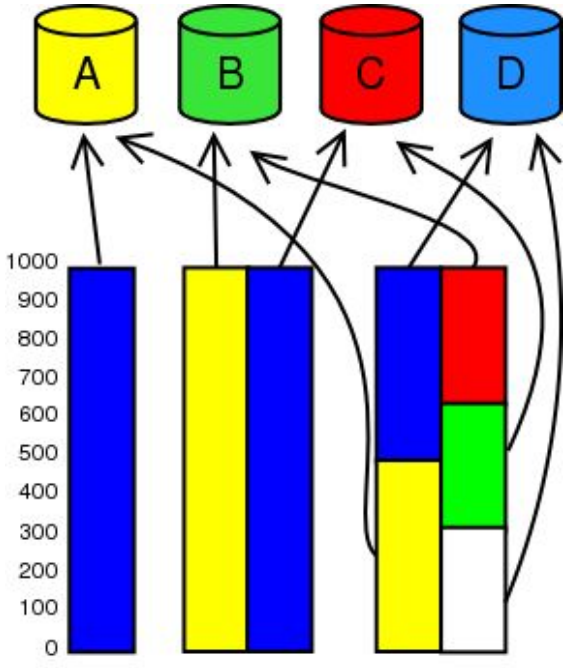

Figura 4-4.4 Exemplos de armazenamento do *exNode* no *IBP server* com 3 estratégias diferentes de replicação

Como visto na figura 4-4 o *exNode* além de permitir a replicação dos dados, ele não restringe o número de réplicas, e essas réplicas não necessitam ser cópias completas dos dados, outra característica importante é que ele pode ser passado para outros clientes.
O ponto chave sobre o *design* do *exNode* é que ele nos permite criar abstrações de armazenamento com fortes propriedades, como um arquivo em rede, que pode ser adaptado ao esquema de comandos sobre o modelo *IBP* de armazenamento de uma maneira totalmente consistente com a abordagem de exposição de recursos.
O *exNode* é expresso concretamente como uma codificação de recursos de armazenamento (*IBP capabilities*) e associado a um metadados em *XML*. Como *IBP capabilities*, essa serialização pode ser passada de cliente a cliente, permitindo um grau de flexibilidade e comportamento do armazenamento em rede. O uso do *exNode* por várias aplicações provê interoperabilidade similar a ser anexado ao mesmo sistema de arquivos de rede [13].

A biblioteca *exNode* possui um conjunto de chamadas (tabela abaixo) que permitem a uma aplicação tanto criar como destruir um *exNode*, adicionar ou apagar um mapeamento

para ele, pesquisá-lo com um conjunto de critérios, e para produzir uma serialização *XML*, para permitir o uso de ferramentas baseadas no modelo *XML* [14].

exNode Management	*Segment Management*
XndCreateExNode **xndFreeExNode**	**xndCreateSegment** **xndFreeSegment** **xndAppendSegment** **xndDeleteSegment**
Segment Query	exNode Serialization
XndQuery **xndEnumNext** **xndFreeEnum**	**XndSerialize** **xndDeserialize**

Tabela 4-4.4 biblioteca conjunto de chamadas *exNode*

Exemplo de arquivo de *exNode* (*xnd*)

```
<?xml version="1.0"?>
...
<exnode:metadata name="filename" type="string">shrike-i386-
disc1.iso</exnode:metadata>
...
<exnode:mapping>
     <exnode:metadata name="alloc_length"
type="integer">22300266</exnode:metadata>
     <exnode:metadata name="alloc_offset"
type="integer">0</exnode:metadata>
     <exnode:metadata name="e2e_blocksize"
type="integer">1048576</exnode:metadata>

     <exnode:metadata name="exnode_offset"
type="integer">646691996</exnode:metadata>
     <exnode:metadata name="logical_length"
type="integer">22299492</exnode:metadata>

     <exnode:function name="checksum">
     <exnode:argument name="blocksize"
type="integer">1048611</exnode:argument>
     <exnode:function name="des_encrypt">
           <exnode:argument name="KEY" type="string">$#-x]/
a</exnode:argument>
           <exnode:argument name="blocksize"
type="integer">1048578</exnode:argument>
     </exnode:function>
```

```
<exnode:read>
     ibp://planetlab2.cs.wisc.edu:6714/IBP-
155031061718223314901067733618113685917786951805196800564220244974891
606744241594543121279638435415923932050068139204389136766652308797627
635530345003/
     655263292/
     READ
</exnode:read>
<exnode:write>
     ibp://planetlab2.cs.wisc.edu:6714/IBP-
155031061718223314901067733618113685917786951805196800564220244974891
606744241594543121279638435415923932050068139204389136766652308797627
635530345003/
     1029278940/
     WRITE
</exnode:write>
<exnode:manage>
     ibp://planetlab2.cs.wisc.edu:6714/IBP-
155031061718223314901067733618113685917786951805196800564220244974891
606744241594543121279638435415923932050068139204389136766652308797627
635530345003/
     655263292/
     READ
</exnode:manage>
</exnode:mapping>
```

4.4 LORS – Logistical Runtime System

Embora o *LBone* torne simples ao usuário encontrar depósitos e o *exNode* possuir *capabilities* para o usuário, o usuário ainda tem que manualmente requerer alocações, armazenar dados, criar o *exNode*, anexar o mapeamento ao *exNode*, e inserir as alocações *IBP* e os metadados nos mapeamentos[16]. A próxima camada na *Network Storage Stack* é o *LoRS*, que é na verdade uma *API* em *C* e uma interface em linha de comando disponíveis para *Linux/Unix/Win32* e *MacOS*, que automatiza a busca por depósitos *IBP*, criando e usando as *IBP capabilities* e criando *exNodes*. O *LoRS* permite ao usuário criar, manipular e utilizar os "*network files*" do *exNode*.

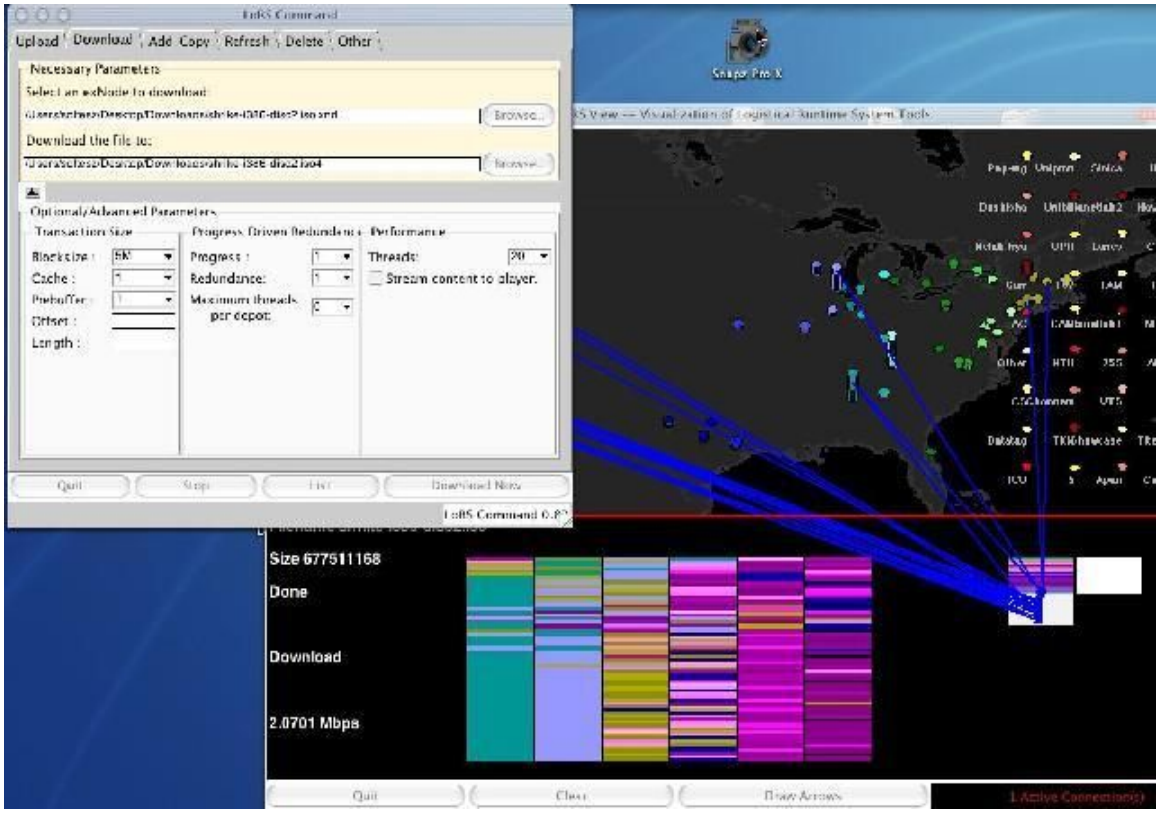

Figura 4-4.5 LoRS

Funcionalidades do *LoRS* :
Upload: Cria arquivo de rede a partir de um arquivo local, *input stream* ou *memory buffer*.
Download: Obtém os *bytes* de um arquivo de rede e o armazena localmente.
Augment: Cria replicas do arquivo de rede.
Trim: Remove replicas do arquivo de rede.
Refresh: Aumenta ou modifica o tempo de vida das *IBP byte arrays*.

Quando o usuário utiliza o *LoRS* ele também pode determinar o número de *threads* (número de conexões), o tamanho do bloco para a transferência do arquivo, e quanta memória utilizar com buffers.

Para proteção dos dados enquanto transitam e enquanto são armazenados no depósito, cada um é considerado um "servidor não confiável", o *LoRS* provê múltiplos tipos de encriptação, incluindo *DES*. Com a *API*, a aplicação pode usar algoritmos adicionais de encriptação e adicionar o tipo de algoritmo e chave como funções de metadados para o *exNode*.

4.4.1 webLoRS

Você também pode usar o *LoRS* sem instalar nada. Esta ferramenta é uma versão limitada da versão linha de comando. Em virtude das ferramentas web utilizarem o protocolo *HTTP* para enviar e receber o arquivo inteiro, ela funciona melhor com arquivos com tamanho até 50 MB.

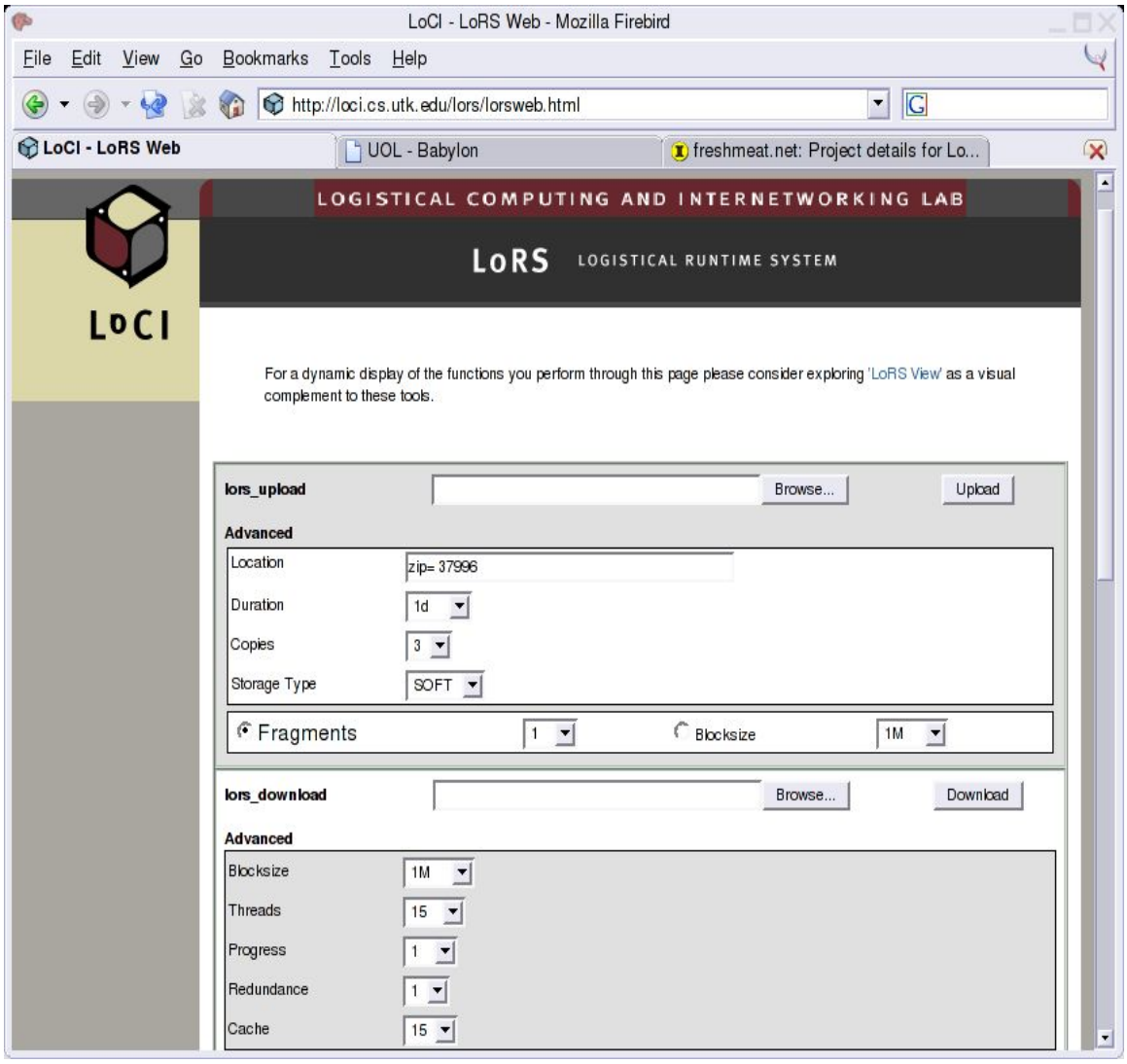

Figura 4-4.6 webLoRS

5 Projetos utilizando componentes do Logistical Network Stack

Até o prezado momento, constatamos que vários projetos estão utilizando a infra-estrutura fornecida pelo *Logistical Network Stack*. Para uma melhor compreensão das possibilidades de aplicações do *Logistical Network Stack*, descreveremos alguns projetos que o utilizam, sendo que entre estes projetos selecionamos dois: o ***NetSolve*** e o ***Video IBPster***, os quais serão melhor detalhados no estudo de casos de aplicação nos próximos capítulos, para que o leitor possa melhor visualizar como é utilizada a infra-estrutura fornecida pelo *Logistical Network Stack*.

- **TAMANOIR.**

O projeto *Tamanoir* é um *framework*, baseado no conceito de *Active Network* (conceito que permite introduzir módulos personalizados nos equipamentos de rede), onde uma nova geração de equipamentos de rede, como roteadores, *proxies* e *gateways*, fornecem serviços para tratar os pacotes de dados, podendo trabalhar os dados em um nível inferior, fazendo: *packets marking*, descarte seletivo, ou em um nível mais alto, fazendo: *QoS*, criptografia, compressão dos dados [8]

- **The Network Weather Service.**

The *Network Weather Service* é um sistema distribuído que monitora e distribui os dados de desempenho de vários recursos computacionais e de rede (sensores de desempenho), sendo desenvolvido para "*schedulers*" dinâmicos e para prover estatísticas de *QoS* sobre o ambiente de rede [17].

- **Ibiblio.**

Ibiblio é um biblioteca livre de informações, que inclui: software, musica, livros literários, de arte, historia, ciência, política, etc, criado por uma associação do *Center for Public Domain* e *The University of North Carolina*. *Ibiblio* utiliza a ferramenta *I2-DSI* para transmissões de dados entre seus servidores. O *I2-DSI* (*Internet2 Distributed Storage Infrastructure project*) é um sistema que utiliza a infra-estrutura *IBP* [10].

- **TerraVision.**

TerraVision sistema iterativo *3D browser* para representação de grandes áreas geográficas, podendo lidar com grande volume de dados obtidos de localidades remotas, incluindo imagens de satélite, dados topográficos, climáticos, construções. Os dados podem ter tamanho de *terabytes*, sendo distribuídos em múltiplos servidores através da *Web* [11].

- **Logistical Distribution Network.**

Este projeto é um sistema para distribuição de conteúdo, como exemplo de conteúdo disponíveis temos as imagens *ISO d*os cds das distribuições *Linux* (*RedHat, Fedora, Debian* e *FreeBSD)*. As imagens dos cds das distribuições são armazenados no *Lbone* e os arquivos de *exNode* são disponibilizados no site do projeto [18].

- **IBPvo.**

IBPvo é um sistema de *web-VCR*, que grava programas da *TV* a cabo e envia o *exNode* para o usuário via *e-mail* [6].

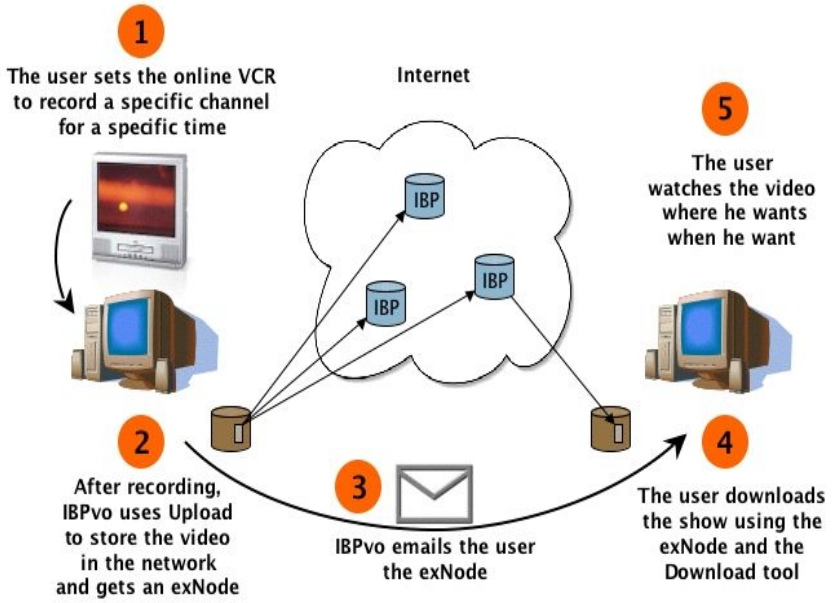

Figura 5-5.1 IBPvo

- **LoRS-Mail.**

O *LoRS-Mail* (*IBP-Mail*) é um sistema de e-mail que permite os usuários enviarem um grande volume de dados para outros usuários, utilizando o *LBone* para armazenar os

dados e enviando o *exNode* anexado ao *e-mail* para o destinatário, que poderá fazer o *download* destes posteriormente [6]

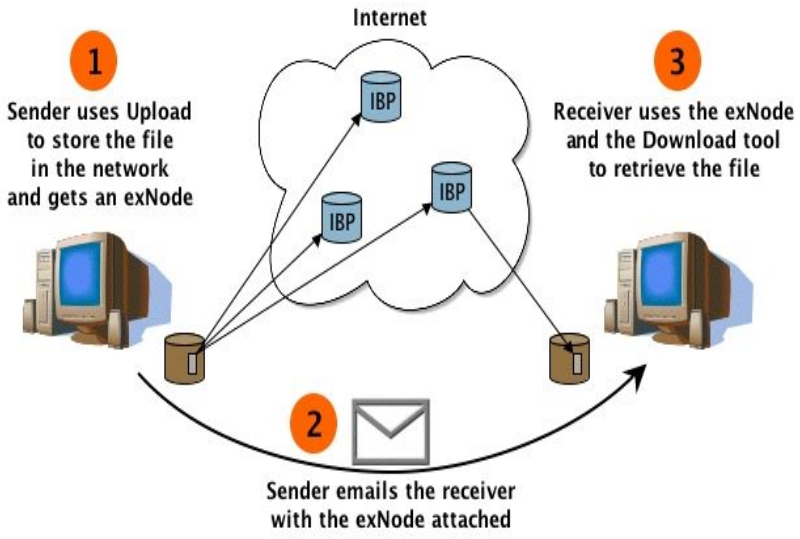

Figura 5-5.2 LoRS-Mail

- **ROLFS.**

ROLFS (*The Read-Only Logistical File System*) é um sistema similar a um sistema de arquivos para *exNodes*, prove meios para compartilhar *exNodes* via *namespacing* (diretórios), *user/group* autenticação/autorização, permissões de arquivo *UNIX-like* [6]. O sistema ROLFS fornece ferramentas para o gerenciamento dos dados similares as utilizadas por sistemas de arquivos tradicionais, como:

rolfs_add_user – adiciona um novo usuário.
rolfs_chgrp – muda o grupo de um arquivo.
rolfs_chmod – muda as permissões de um arquivo.
rolfs_chown – muda o proprietário de um arquivo.
rolfs_extract - recupera um arquivo no servidor ROLFS.
rolfs_inject – adiciona um arquivo no servidor ROLFS.
rolfs_ls – lista arquivos e diretórios.
rolfs_mkdir – cria um novo diretório.
rolfs_rm – remove um arquivo do servidor ROLFS.
e rolfs_rmdir – remove um diretório.

6 Estudo de caso

Na descrição do projeto *NetSolve/GridSolve* procuramos enfatizar a arquitetura do sistema e no projeto *Video IBPster* nos testes de desempenho, pois este utiliza todos os componentes do *Logistical Network Stack*, como descritos anteriormente.

6.1 NetSolve/GridSolve

O *NetSolve/GridSolve* é um sistema cliente/agente/servidor que habilita os usuários resolverem remotamente problemas científicos complexos. O propósito do *NetSolve/GridSolve* é criar o *middleware* necessário para prover um conexão entre interfaces padrões de programação, o *Desktop Scientific Computing Environments* (*SCEs*) com os serviços suportados pela arquitetura *Grid* [7].

Características:

- *RPC*.
- Interfaces: *Fortran, C, Matlab, Mathematica*
- Flexibilidade para incorporar novos componentes.
- Facilidade de uso.

O *NetSolve/GridSolve* é um exemplo de utilização dos componentes do *Logistical Network Stack*, e como estes componentes se encaixa na arquitetura do *NetSolve/GridSolve*. Neste sistema o *IBPserver* é utilizado como *cache* de comunicação para o servidor *NetSolve/GridSolve*, principalmente quando este necessita enviar os dados para múltiplos clientes. Na figura 6-1 podemos ver como é composta a arquitetura do sistema *NetSolve/GridSolve*, formada pelo Cliente e Servidor *NetSolve/GridSolve* e o Servidor *IBP*, também podemos ver como estes interagem.

Os desenvolvedores do *NetSolve/GridSolve* pretendem adicionar outros elementos do *Network Logistical Stack* na arquitetura *NetSolve/GridSolve*, para ampliar os recursos oferecidos pelo sistema.

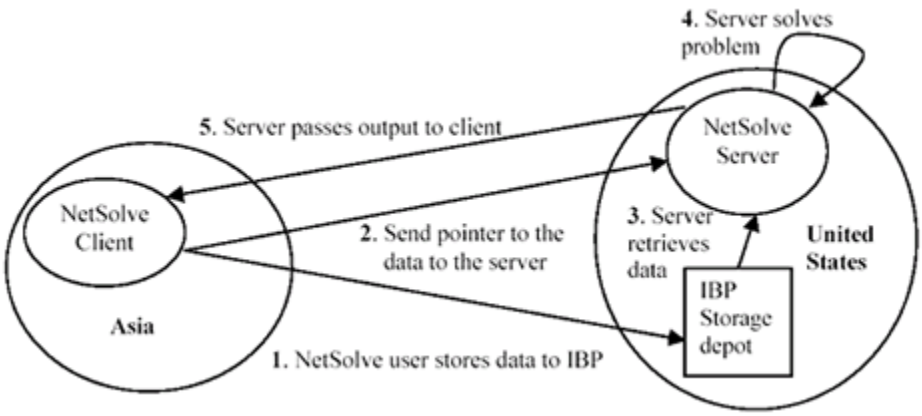

Figura 6-6.1 Estrutura NetSolve com IBPserver

6.2 Video IBPster

Pesquisadores do *LoCI* (*Logistical Computing and Internetworking)* na Universidade do Tennessee, onde o trabalho sobre *Logistical Networkin*g foi iniciado, desenvolveram uma aplicação muito simples - o *Video IBPster* - para demonstrar como esta tecnologia pode ser empregada para executar melhor e ter mais flexibilidade para distribuir arquivos.

O *Video IBPster* é uma de aplicação que pode distribuir vídeo com alto desempenho usando tecnologia mais simples e de baixo custo para distribuição de video. Para alcançar este objetivo são combinados todos os componentes básicos do *Logistical Networking - Internet Backplane Protocol (IBP)*, o *exNode*, o *Logistical Runtime System (LoRS)*, e o *Logistical Backbone (L-Bone).* A **figura 6-2** a seguir são mostrados o *Video IBPster* em funcionamento. Os componentes observados são todos baseados em *software* livre e o *download* é gratuito a partir do repositório *LoCI*, também encontra-se disponível toda a documentação.

A demonstração do *IBPster* combina a linha de comando do *LoRS* e software livre de áudio e programas de vídeos. Para demonstração, são usadas duas interfaces gráficas de usuário do tipo *TCL/TK (GUI).* Uma faz o controle da linha de comando da ferramenta *LoRS* e a outra é um mapa, que representa o *LBone* com a localização dos depósitos do *IBP* (figura 6-2).

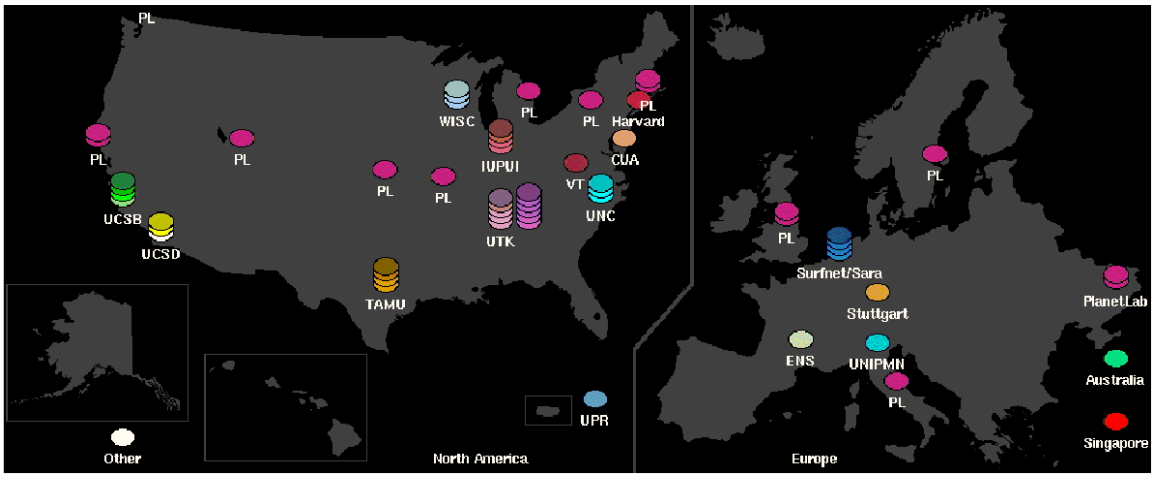

Figura 6-6.2 Lbone map with IBP depots

Primeiro o usuário faz um *upload* do arquivo e cria um *exNode* (figura 6-3). Neste exemplo, a replica foi colocada no *UNC* e outra foi colocada no *UCSB* (figura 6-4). Note que todos os dez pedaços estão sendo copiados simultaneamente. Atualmente, a ferramenta não tenta determinar qual réplica ira ter o melhor desempenho como a fonte da cópia. Ainda esta sendo realizados pesquisas sobre esta questão.

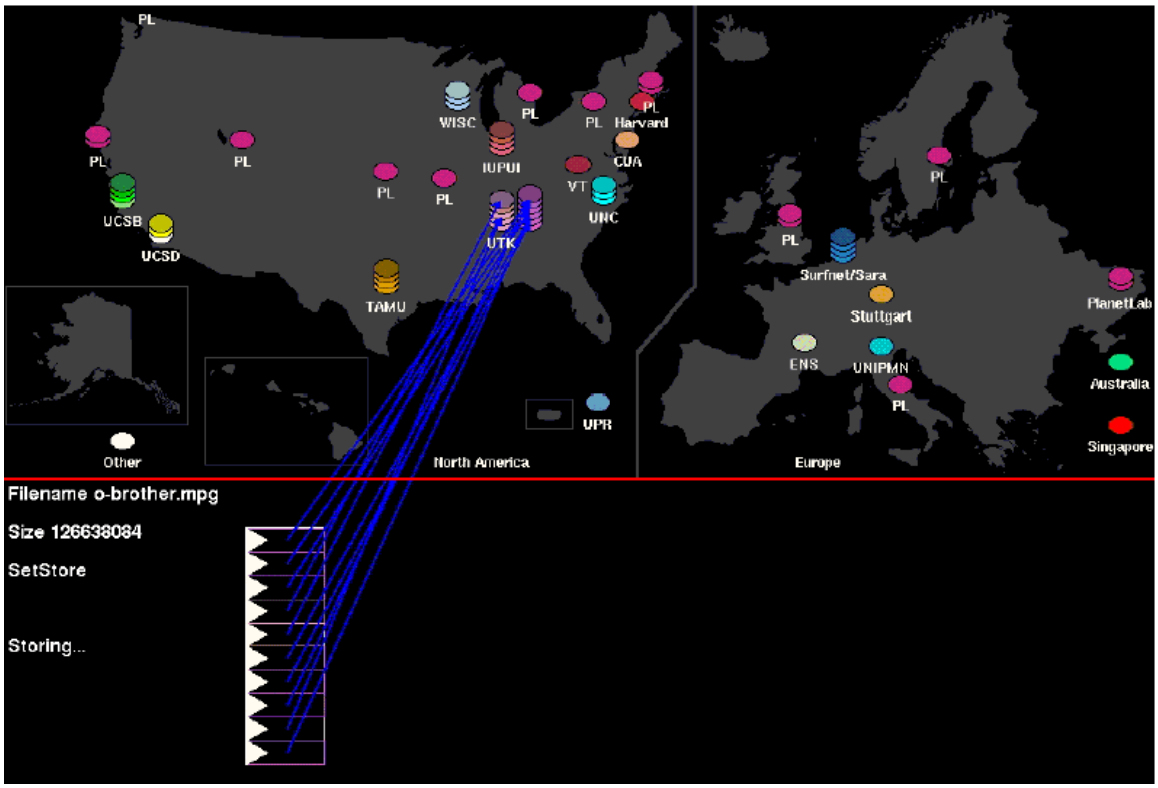

Figura 6-6.3 Upload in progress

Por ultimo o usuário pode fazer o download do arquivo para o disco ou direto na ferramenta multimídia capaz de ler dados a partir do padrão de entrada (*stdin*) como por exemplo: (*e.g. mplayer, mpg123, VideoLan Client, etc*). Neste exemplo foram armazenados arquivos de 126 MB (figura 6-5). A ferramenta de download esta fazendo 8 download paralelos de blocos de 2 MB a partir de vários depósitos no *UTK*. Os blocos estão sendo carregados e são representados por retângulos vazios, mais abaixo a direita da figura. Após o download dos dados, o bloco é preenchido. A coluna branca mais embaixo à direita representa os blocos de dados que foram feito o *download* e atualizados no disco ou multimídia *player*.

Durante a demonstração, foram transferidos em media 15 *Megabit/s* (*Mbps*) de vídeo a partir dos Estados Unidos. As ferramentas do *LoRS* foram projetadas para transferir arquivos de dados, não apenas multimídia, sendo usado o protocolo *TCP* para transferência. Por causa disto, *IBPster* não interrompe frames. Se a ferramenta usada para *download* não envia dados suficientes para o *mplayer*, o áudio espera enquanto recebe mais dados. Se isto acontece varias vezes seguidas, o *player* começa a falhar. Acredita-se que de download de 10-15 threads baixando blocos de 512 KB a partir de 8-10 depósitos de *IBP* devem enviar dados para o *player* mais rápido o suficiente para 15 Mbps de vídeo.

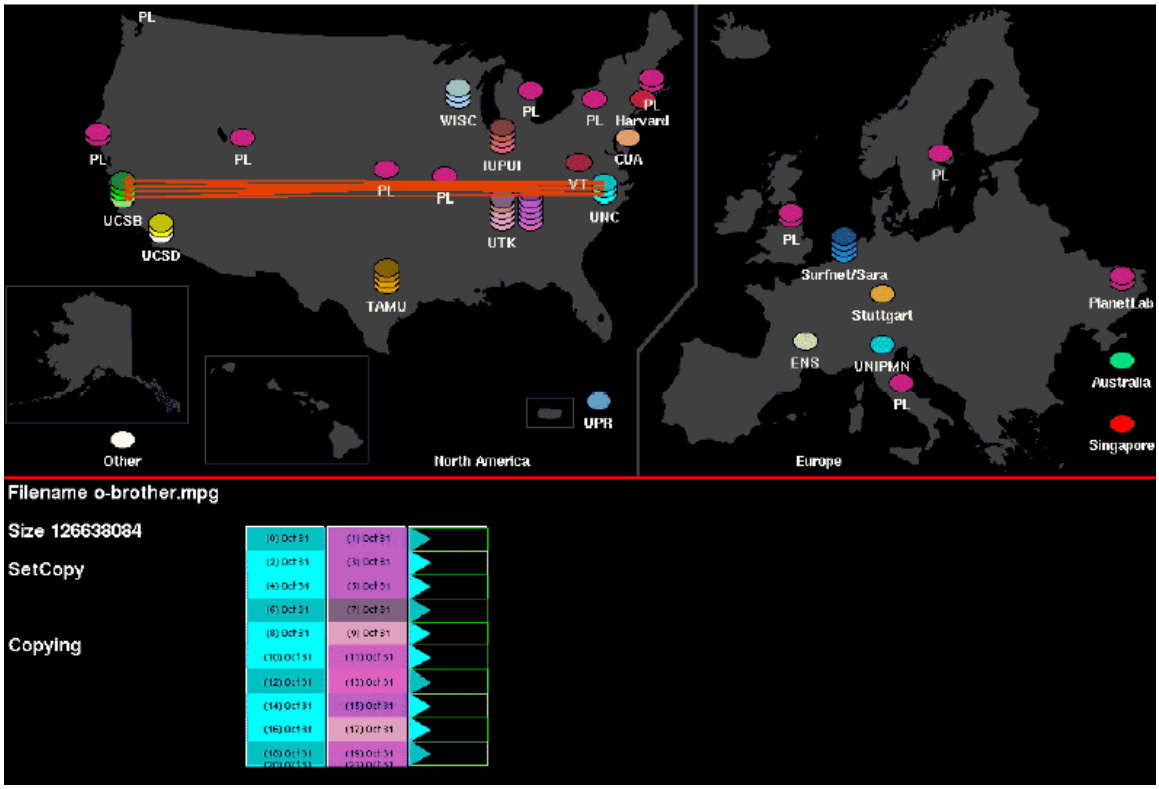

Figura 6-6.4 Augment adding replica in UCSB

Neste exemplo foram armazenados arquivos de 126 MB (figura 6-5). O ferramenta de *download* esta fazendo 8 *download* paralelos de blocos de 2 MB a partir de vários depósitos no *UTK*. Os blocos que estão sendo carregados, são representados por retângulos vazios, mais abaixo a direita da figura. Após o *download* dos dados, o bloco é preenchido. A coluna branca mais embaixo à direita representa os blocos de dados que foram feito o *download* e atualizados no disco ou multimídia *player*.

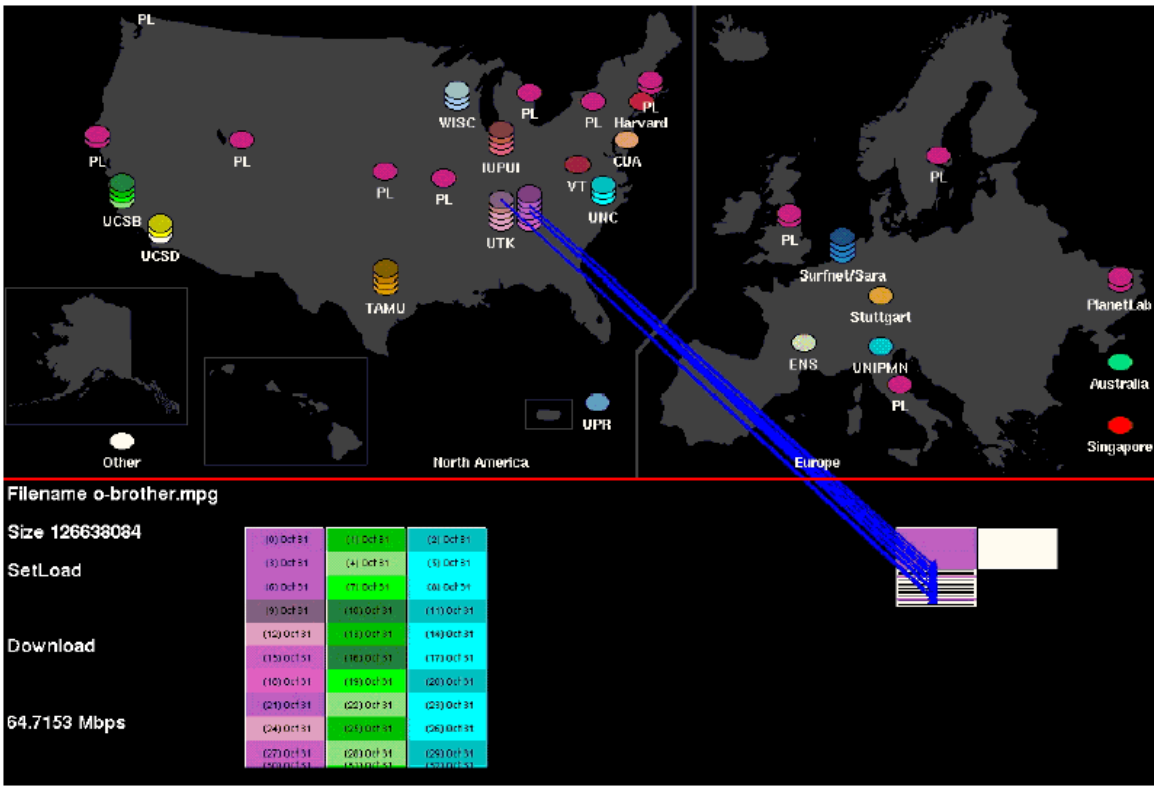

Figura 6-6.5 Downloading 2 MB blocks in parallel

A seguir são mostrados três gráficos com os resultados dos testes realizados.

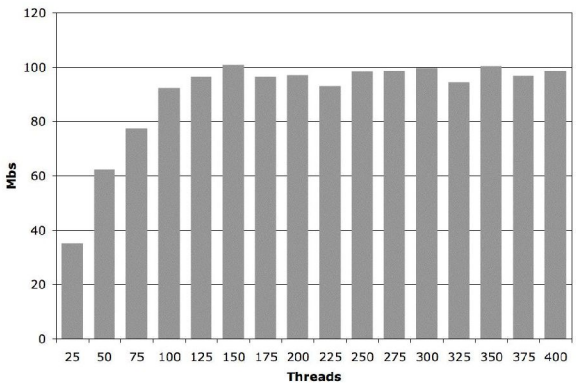

Figura 6-6.6 Video IBPster Teste 1 - US.

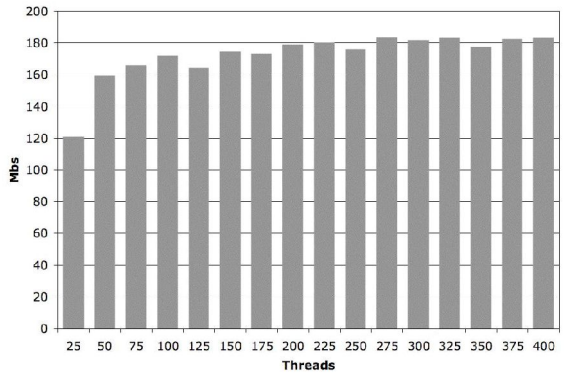

Figura 6-6.7 Video IBPster Teste 2 – Europa

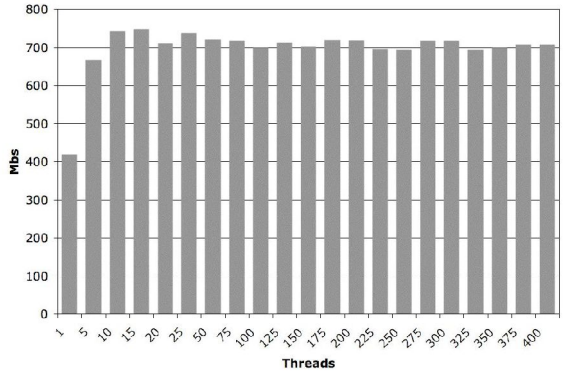

Figura 6-6.8 Video IBPster Teste 3 - Sara

7 Conclusão

A proposta feita pelos membros dos Laboratórios *LoCI* para utilização do *Logistical Network Stack* como infra-estrutura para *Data Grid* mostra-se audaciosa pois propõe mudanças no paradigma dos sistemas de armazenamento e comunicação e uma mudança no modelos de desenvolvimento, normalmente centrado nos computadores, para um modelo centrado na internet. Também foi proposto que o desenvolvimento seja aberto, o que já é comum no meio acadêmico.

Porém, um ponto que devemos salientar é o referente a segurança e a proteção contra abusos; a proteção contra abusos pode tornar-se um problema caso os administradores dos servidores *IBP* não adotarem uma política adequada sobre a utilização dos recursos, porém este problema pode ser mais facilmente contornado, se comparado ao problema de se gerenciar as credenciais de acesso (*Capabilites*) aos dados pelos usuários, pois como a proteção contra o acesso indevido as *capabilites* ficam totalmente ao encargo do usuário, o qual pode não tomar as devidas precauções, o que pode comprometer a integridade e confidencialidade dos dados; caso as *capabilites* sejam apropriadas por pessoas não autorizadas.

Outra proposta, feita pelo *LoCi*, é que os recursos fornecidos pelos *DataGrids* não sejam exclusivos os membros de comunidades virtuais que possuam seu próprio *grid*, mas sim acessíveis também por todos na internet, para isto, criaram um *DataGrid*, chamado *LBone*, que conta com mais de 10 TB disponíveis para toda a comunidade através de aplicativos clientes *IBP*.

8 Bibliografia

[1] M. Beck, T. Moore, J. S. Plank, M. Swany, Logistical Networking; Innvation Computing Laboratory, Department of Computer Science, University of Tennessee

[2] A. Bassi, M. Beck, T. Moore, J. S. Plank; The Logistical Backbone: Scalable Infrastructure for Global Data Grids; LoCI Laboratory, University of Tennesse.

[3] S. Atchley, S. Soltesz, J. S. Plank, M. Beck; Video IBPster; Logistical Computing and Internetworking Lab., University of Tennesse.

[4] A. Bassi, M. Beck, J. S. Plank, R. Wolski; Internet Backplane Protocol: API 1.0; Department of Computer Science University of Tennessee.

[5] A. Bassi, M. Beck, E. Fuentes, T. Moore, J. S. Plank; Logistical Storage Resources for the Grid; LoCI Laboratory, University of Tennesse.

[6] LoCI; http://loci.cs.utk.edu

[7] Netsolve; http://icl.cs.utk.edu/netsolve/

[8] Tamonoir; http://lhpca.univ-lyon1.fr/~tamanoir/

[9] Open Video; http://www.unc.edu/

[10] iBiblio; http://www.ibiblio.org/

[11] Terra Vision; http://www.tvgeo.com/index.shtml

[12] M. Beck, T. Moore, and J. Plank. An end-to-end approach to globally scalable network storage. In ACM SIGCOMM 20002 Conference, Pittsburgh, PA, USA, August 2002.

[13] S. Atchley, S. Soltesz, J. S. Plank, M. Beck, and T. Moore. Fault-tolerance in the network storage stack. In IEEE Workshop on Fault-Tolerant Parallel and Distribuited Systems, Ft. Lauderdale, FL April 2002.

[14] B. Alessandro, M. Beck, G Fagg, T. Moore, J. S. Plank, M. Swany, R. Wolski. The Internet Backplane Protocol: A Study in Resource Sharing.

[15] B. Alessandro, M. Beck, T. Moore, J. S. Plank. The Logistical Backbone: Scalable Infraestructure for Global Data Grids.

[16] S. Atchley, M. Beck, T. Moore, J. S. Plank, J. Millar, S. Soltesz. The Logistical Networking Testbed.

[17] The Network weather service; http://nws.cs.ucsb.edu/

[18] Logisical Distribution Network; http://promise.sinrg.cs.utk.edu/lodn/

[19] Baird, I - "Understanding Grid Computing", In: http://www.platform.com/pdfs/whitepapers/understanding_grid.pdf

[20] GriPhyN - Education & Outreach Center, In: http://www.phys.utb.edu/griphyn/

[21] Laszewski, G. V. – "Gregor von Laszewski – Grid Course", In: http://www-fp.mcs.anl.gov/~gregor/grid-iit/talks/Introduction%20to%20Grid%20Computing.htm