Guia do Usuario do VOID3

Versao 0.1

Sumario

1 Introducao

O VOID e um *framework* que permite descrever e implementar aplicacoes como um gra co de componentes, que podem executados em um ambiente heterogêneo distribu do. O modelo de programacao do VOID, chamado de *Iter-stream*, representa componentes de aplicacoes intensivas em dados como um conjunto de ltros, como visto na gura 1. Cada componente opera no estilo de uxo de dados, onde repetidamente lê de *bu ers* de entrada (*streams*), executa o processamento de nido pela aplicacao, sobre os dados lidos, e escreve em algum *stream* de sa da. O VOID provê um conjunto basico de servicos, na sua maioria referentes a como1e7(a)-3.(V)27(OID3)]TeJ

eler o arquivode con guracaodo sistema, que edescrito na secao4.2, garantirque as aplicacoes sejam executadas de acordo com o especi cado e cuidar da monitoracao das aplicacoes em execucao. Os ltros, por sua vez, percorrem um caminho diferente, antes de comecarem a execucao esses recebem, enviadas pelo gerente, informacoes necessarias a suas con guracoes, estas informacoes sao, dentre outras:

O nome do ltro;

Quantos ltros do seus tipo existem;

```
#!/bin/bash

#Directory do programa
DIR_PROG="/home/speed/george/void3/samples/sample_print_task"

# Coloque aqui o directory contendo as bibliotecas do Datasucker
#e dos filtdos
DIR_VOID=../../
DIR_FILTROS=./

DISPLAY=orfeu:0

# se quiser depurar nao comente
if [ `hostname` = "eirene" ]
then
        DEBUG=1
fi
```

4.2.1 *HostDec*

No *hostdec* declara-se todas maquinas dispon veis para execuacao da aplicacao, tambem podemos, nesta secao, associar a cada *host* recursos e quantidade de memoria. Isso facilita a vida do desenvolvedor, pois o mesmos nao nem precisa saber em quais maquinas a aplicacao esta sendo executada, alem disso, e possivel associar recursos a uma maquina e no *placement* o programador informa que determinado Itro depende daquele recurso, portanto deve rodar no local no qual o mesmo esta disponivel.

4.2 Conf.xml

```
        </host>
</hostdec>

<placement>
        <filter name="filterA">
                <instance demands="bla"/>
        </filter>
        <filter name="filterB"/>
</placement>

<layout>
        <stream>
```

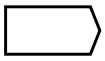

Figura 3: DivP4

5.2.1 Inicializacao

A funcao de inicializacao (*initFilter*) e chamanda sempre que chega uma nova unidade de trabalho. Isso permite que os ltros solicitem recursos necessarios, tais como pegar suas portas de comunicacao, abrir arquivos, etc. O prototipo dessa funcao e o seguinte:

*int initFilter(FilterArg *arg)*

O argumento do tipo *FilterArg* contem um void* que aponta para o *work* e um *int*, onde e armazenado o tamanho desse work em *bytes*. A seguir temos o exemplo de *init* do ltro divP4:

```
int initFilter(FilterArg *fa){
       //pega os handlers de saida
        dividendoP      = dsGetInputPortByName("dividendoP");
        restoP          = dsGetOutputPortByName("resto");
        quocienteP      = dsGetOutputPortByName("quociente");

        return 1;
}
```

Neste exemplo apenas pegamos os *handlers* das portas de sa da e entrada do ltro. Poderimos nesse local utilizar o *work*, mas isso nao e feito.

5.2.2 Processamento

Quando o ltro retorna do *initFilter* o *processFilter* e chamado tendo como argumento o tendo como parametro o *FilterArg*, assim como no *initFilter*. O prototipo dessa funcao e o seguinte:

*int processFilter(FilterArg *fa)*

No exemplo de *process* do divP4, abaixo, o filtro lê um inteiro da entrada divi-

5.3 Tolerância a Falhas

O modelo de tolerância a falhas do VOID3, em implementacao, permite ao programador dividir sua aplicacao em tarefas. Onde uma tarefa e um evento global com in cio e m bem de nidos em cada processador, embora execute assincronamente em cada deles. Ela pode ler dados de tarefas anteriores ou gravar dados para serem usados por outras tarefas. Assim a tarefa deve possuir um identi cador, cuja unicidade e as dependências[2] sao garantidas pela aplicacao.

Nesse modelo nao sao permitidas mensagens entre tarefas, a dado516(unica)-471 simultaneamente, desde que nao exista dependência de dados entre elas. O que e

5.4 Criacao da Aplicacao Principal

O \main"da aplicacao, como pode ser visto no exemplo abaixo, inicia-se com uma chamada a funcao *initDs*. Nessa funcao a aplicacao, caso for o gerente, lê o arquivo conf.xml, descrito na secao 4.2, inicia os filtros e envia os dados de configuracao para os mesmos. Se forem filtros, logicamente, os mesmos ficam aguardando os dados de configuracao.

int getData(int taskId, char *id, void **val): Aponta o val para os dados armazenados cujo identi cador e *id* e a tarefa e *taskId*.

int removeData(int taskId, char *id): Remove os dados armazenados cujo identi cador e *id* e a tarefa e *taskId*

7.1 DivP4

```
}fTJ -18. 0 -88.89 char15(=)-*confFilejobs =
```

```
      //le o dividendo
      int num_jobs = ((int *)getFAWork(fa))[0];

0eturn 1;
      dsWriteBuffer(numP, &num_jobs, sizeof(int));                                    }

      return 1;
    }f(int));




    return 1;
  }f]TJ ET 0.9 m 7227.4 -250.19 m 7227.4 -36.69 l S 0.4 w 27..4 -3 0.66 m 452..4 -3 (
```

```
divP4.so: divP4.c
   ${CC} ${CFLAGS} ${CLIBS} -shared -o divP4.so divP4.c

imp.so: imp.c
   ${CC} ${CFLAGS} ${CLIBS} -shared -o imp.so imp.c

leNum.so: leNum.c
   ${CC} ${CFLAGS} ${CLIBS} -shared -o leNum.so leNum.c

main: main.c
   ${CC} ${CFLAGS} ${CLIBS} main.c -o main

clean:
   rm -f *.o divP4.so imp.so leNum.so main
```

conf.xml

```
config>
<hostdec>
   <host name="eirene" mem="512M">
      <resource name="bla" />
   </host>
   <host name="orfeu"/>

</hostdec>
<placement>
   <filter name="leNum" libname="leNum.so" instances="1">
      <instance demands="bla"/>
   </filter>
   <filter name="divP4" libname="divP4.so">
   </filter>
   <filter name="impResto" libname="imp.so" />
   <filter name="impQuociente" libname="imp.so"/>
</placement>
<layout>
   <stream>
      <from filter="leNum" port="saida"  />
      <to filter="divP4" port="dividendoP"  />
   </stream>
   <stream>
      <from filter="divP4" port="resto"  />
      <to filter="impResto" port="entrada"  />
```

7.2 Sample

```
    // Pega o Work
    numToPrint = (int *) getFAWork(arg);

    // Imprime para a saida de erro
    fprintf(stderr, "Inicializing filter A\n");
```

```
        // guarda dados da ultima tarefa criada
        putData(oldTaskId, id, val, VAL_SIZE);

        // cria tarefa que depende da tarefa criada anteriormente.
```

```
   return 0;
}
```

IterB.c

```
#include <stdio.h>
#include <stdlib.h>
#include "filterB.h"
#include <unistd.h>

InputPortHandler input;
OutputPortHandler output;

int initFilter(FilterArg *arg) {
```

*7.2 Sample_*