Implementação de um Balanceador de Carga para a Biblioteca AMPI Baseado no Modelo de Escalonamento MigBSP

Roberto de Quadros Gomes[1], **Rodrigo da Rosa Righi**[1]

[1]Programa Interdisciplinar de Pós-Graduação em Computação Aplicada-UNISINOS
Av. Unisinos,950 – 93.022-000 – São Leopoldo – RS – Brasil

rqg.gomes@gmail.com, rrrighi@unisinos.br

Resumo. *Este artigo descreve os primeiros passos para a implementação de um balanceador de carga, segundo o modelo MigBSP, para a biblioteca AMPI. Ele apresenta como serão adquiridos as métricas utilizadas pelo escalonador e como será feita o processo de migração de fato. O objetivo deste trabalho é fornecer ao programador este modelo de balanceamento de carga sem custos de programação.*

1. Introdução

Migração de processos é uma técnica pertinente ao remapeamento de um processo para um processador mais rápido ou aproximá-lo de outros os quais se comunica frequentemente. Em particular, a biblioteca AMPI (Adaptive Message Passing Interface) [Huang et al. 2004] oferece um arcabouço uniforme para balanceamento de carga de forma transparente para o usuário. Além disto, a AMPI se destaca pela velocidade na tarefa de migração uma vez que faz uso da abstração de processadores virtuais. Esse artigo descreve os primeiros passos para o desenvolvimento de um balanceador de carga para AMPI que segue o modelo de escalonamento MigBSP [Da Rosa Righi et al. 2010]. MigBSP atua sobre aplicações em fases do tipo BSP (Bulk Synchronous Parallel), onde as migrações são viabilizadas entre o fim de uma superetapa e o início da próxima. Ele combina dados de métricas tais como processamento, comunicação e memória para o cálculo da sua função de decisão chamada Potencial de migração (PM). PM possui os parâmetros i e j que denotam um processo e um processador destino, respectivamente. Em suma, ele é atingido da seguinte forma: $PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j)$. O balanceador proposto é denominado MigBSPLB.

2. Passos Iniciais para o Desenvolvimento de MigBSPLB

MigBSPLB implementará todas as questões respondidas por MigBSP, que são: (i)"Quando" tratar a migração de processos; (ii) "Quais" processos são candidatos a migração; e (iii) para "Onde" migrar os processos selecionados entre os candidatos. A implementação das três sentenças acima passa pela captura e tratamento de múltiplas métricas para as tomadas de decisão. Além das métricas habituais como carga e comunicação entre processos, um diferencial que o modelo MigBSP apresenta é o cálculo de custo de migracão a ser realizado. Isto é efetuado considerando a memória do processo em bytes vezes o tempo de transferência de um byte entre dois processadores do ambiente. Este parâmetro é representado pela parcela Mem no cálculo de PM. Também será dada atenção a implementação de adaptações que irão gerenciar o intervalo mais propício de superetapas para lançar o reescalonamento de processos. MigBSPLB será escrito

com as diretivas definidas pelo middleware *Charm++*, que é usado como substrato de comunicação pela AMPI. Para a captura das informações, MigBSPLB fará uso de três classes: *ProcArray*, *ObjGraph* e *LDCommData*. Estas classes serão inicializadas a partir do objeto do tipo *LDStats* fornecido pelo arcabouço de balanceamento de carga *Charm++*. *LDStats* possui informações sobre os processadores, processos e comunicação entre eles. *ProcArray* é um vetor de objetos de classes *ProcInfo*. Estes objetos contém dados de velocidade, identificação, disponibilidade e carga de processamento de processadores físicos. *ObjGraph* possui as informações da rede dos processos informando o grafo de comunicações do sistema. Esta classe permite determinar em qual processador um objeto nó está através do método $getCurrentPe()$ e indicar para qual processador o processo deve ser migrado com $setNewPe(proc\_index)$. O método $convertDecisions(LDStats*)$ transmite ao arcabouço todas as alterações dos processos a serem migrados. A classe *LDCommData* informa os dados de comunicação. A estrutura padrão de um balanceador de carga já é definida pela classe *BaseLB*. A implementação principal deste trabalho se dará no método work(LDStats*). A Figura 1 ilustra uma organizaçao inicial da implementação desse método.

```
void MigBSPLB::work(LDStats *stats)
{
  ProcArray *parr = new ProcArray(stats);//dados dos processadores
  ObjGraph *ogr = new ObjGraph(stats);//dados dos objetos
  LDCommData comm[ogr->vertex.size()];//dados de comunicação
  MigBSPComputePM(parr,ogr,comm);       //Estratégia de Migração
  MigBSPmigration(ogr);                 //MigBSP
  ogr->convertDecisions(stats); //envia para o arcabouço informaçoes de migração
}
```

Figura 1. Trecho de código para implementação do MigBSPLB

O método $MigBSPComputePM$ irá gerar o pontencial de migração dos processos existentes. Já o método $MigBSPmigration$ avaliará e realizará a migração dos processos necessários.

3. Conclusão e Trabalhos Futuros

Com a execução deste trabalho, o modelo de escalonamento MigBSP terá uma implementação funcional em linguagem C++. Trabalhos futuros compreendem a execução do MigBSPLB com uma aplicação de compressão de imagens utilizando a técnica de fractais no *cluster* da UNISINOS. A evolução natural dos testes ocorrerá através de contato com o grupo de pesquisa da AMPI para execução de aplicações mais pesadas para a validação do balanceador apresentado.

Referências

Da Rosa Righi, R., Pilla, L. L., Maillard, N., Carissimi, A., and Navaux, P. O. A. (2010). Observing the Impact of Multiple Metrics and Runtime Adaptations on Bsp Process Rescheduling. *Parallel Processing Letters*, 20(02):123–144.

Huang, C., Lawlor, O., and Kale, L. (2004). Adaptive MPI. In Rauchwerger, L., editor, *Languages and Compilers for Parallel Computing*, volume 2958 of *Lecture Notes in Computer Science*, pages 306–322. Springer Berlin Heidelberg.