MCMPI: uma biblioteca com elasticidade para ambientes com múltiplos domínios e nuvem pública

Carlos A. T. Aguni, Liria M. Sato, Edson T. Midorikawa

[1] Escola Politécnica da Universidade de São Paulo

{carlos.aguni,liria.sato,emidorik}@usp.br

Resumo. *Este artigo apresenta uma nova biblioteca que utiliza e estende o padrão MPI, capaz de agregar servidores e clusters localizados em múltiplos domínios, como também recursos da nuvem. A plataforma de execução é criada e entregue à aplicação MPI de forma transparente sem que seja necessário recompilar o código. A solução também provê funções de provisionamento, adição e remoção de nós em tempo de execução trazendo elasticidade à aplicação. Através de benchmarks, comparou-se seu desempenho com a execução nativa da aplicação utilizando a Biblioteca MPI. Foi, também, desenvolvido um protótipo de aplicação elástica com resultados otimistas e dentro do esperado.*

1. Introdução

O crescente número de aplicações de natureza escalável e elástica traz a necessidade cada vez maior de um conjunto de máquinas capaz de suportar tais *workloads*. Junto a isso, também vem aumentando a variedade de arquiteturas de máquinas, o que inclui processadores *multi-core* e *manycore*, *clusters* de computadores, tecnologias de rede e a recente entrada da *cloud*, além das diferentes implementações da biblioteca MPI.

Atualmente existe, também, uma variedade de plataformas, provedores, tecnologias e organizações que trazem desafios para os usuários, que enfrentam obstáculos como federação, autenticação e políticas de acesso dificultando a necessidade principal de poder utilizar e extrair o máximo dessas plataformas de forma integrada, como apresenta a Figura 1. Frente ao desafio de se interconectar essas plataformas, soluções que buscam uma melhor experiência para o usuário, ao mesmo tempo garantindo segurança e interoperabilidade entre elas, são desejáveis. Outra necessidade, intensificada com o advento da nuvem, é a possibilidade de prover elasticidade nas aplicações, tornando-as capazes de agregar e remover recursos computacionais dos seus ambientes de execução. Tais recursos podem estar localizados tanto na nuvem quanto fisicamente.

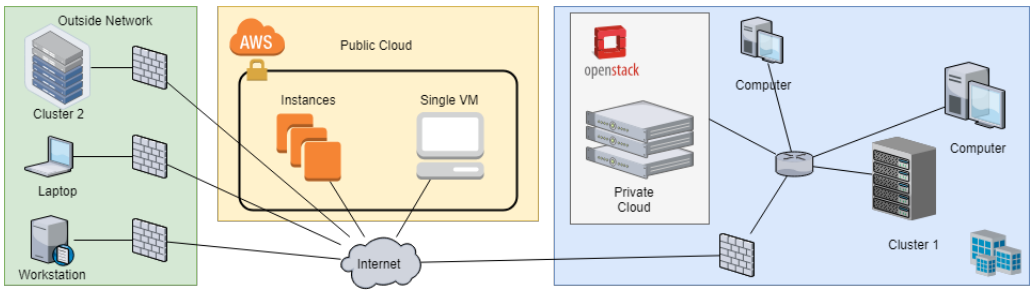


Figura 1: Esquema geral do ambiente de execução da biblioteca MPI proposta.

O presente trabalho tem por objetivo apresentar a proposta e implementação da biblioteca MCMPI (**M**ulti **C**loud/**C**luster **MPI**) que estende a biblioteca MPI oferecendo a capacidade de agregar recursos computacionais, abstraindo a plataforma de execução com servidores e *clusters* localizados em domínios distintos, podendo também adicionar recursos criados sob demanda da nuvem. Preenchendo um arquivo de configuração que representa o ambiente de execução, a biblioteca se encarrega de montar o ambiente e entregar à aplicação sem a necessidade de recompilar o código da aplicação. A biblioteca também dispõe de funções de adição e remoção dinâmica de processos e seus respectivos nós do ambiente de execução corrente, dinamicamente durante o processamento, provendo elasticidade à aplicação. Testes preliminares mostraram que diferenças no desempenho foram imperceptíveis quando comparado com a implementação MPI nativa e foi implementado um protótipo de uma aplicação "cliente $\times$ servidor" a fim de avaliar o comportamento de uma aplicação elástica com resultados que atenderam às expectativas. O artigo encontra-se organizado na seguinte forma: na seção 2 são analisados trabalhos relacionados. A seção 3 descreve os detalhes da biblioteca MCMPI. A seção 4 apresenta os testes realizados. Por fim, a seção 5 apresenta as considerações finais e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

Destaca-se que soluções que buscam implementar *Grid Computing* existem desde a década de 1990 e o avanço da tecnologia permite que novas abordagens sejam exploradas, as quais algumas estão listadas a seguir.

Massetto [Massetto 2007], em sua Tese de Doutorado, comprovou sua solução interligando ambientes distribuídos, compostos de *clusters*, computadores e *workstations* e com sistemas operacionais híbridos, proporcionando um mecanismo transparente de comunicação para o usuário. Utilizou *gateways* localizados na borda de cada rede interna, que devem ser colocados em execução juntamente com a execução da aplicação, como forma de comunicação entre os diferentes domínios. A aplicação utiliza em seu código fonte chamadas equivalentes à MPI específicas para o sistema proposto. Assim, cada mensagem entre máquinas de IPs globais diferentes é então processada e redistribuída pelo *gateway*. Também vale mencionar que a biblioteca MPI utilizada em cada nó é a MPI local já instalada.

Também já foram propostas soluções baseadas em NAT (*Network Address Translation*) [Choi et al. 2004] onde a transmissão de mensagens através da biblioteca MPI sobre ambientes de *grid* é realizada pelo nó *front-end* do *cluster* através de um serviço NAT. O uso de NAT gera um ganho de desempenho uma vez que um pacote já encontra seu destino através da camada de rede não precisando passar pelas camadas seguintes.

[M. Caballer 2021] apresentou a possibilidade de se adicionar *clusters* inteiros à aplicação a partir de recursos localizados em *clusters* físicos e nuvem. Sua solução consiste em criar e/ou designar uma máquina a ser configurada como "vRouter Central Point" na borda de cada domínio por onde os servidores internos o deverão apontar como *gateway* e servidor DHCP (*Dynamic Host Configuration Protocol*). As máquinas *gateway* são conectadas utilizando VPN e, uma vez formada a plataforma de execução, quaisquer aplicações podem ser executadas.

Os trabalhos que mais se assemelham ao trabalho proposto são: [P. Patchin 2009],

[A. Raveendran 2011] e [Dorier et al. 2022]. .

O trabalho *A Framework for Elastic Execution of Existing MPI Programs* [A. Raveendran 2011] procura executar a aplicação utilizando máquinas da AWS (Amazon Web Services) utilizando *templates* de imagens pré-configuradas para criação de novas instâncias. Foi feita a customização do processo `mpd`, responsável por iniciar processos MPI. No momento em que é decidida a mudança no número de máquinas, é feito *checkpoint* da aplicação seguido da recriação total do ambiente com o número atualizado de nós.

Colza [Dorier et al. 2022] introduz aplicações de natureza que as tornam mais complexas devido ao aumento da quantidade de dados no decorrer da execução, demandando mais recursos de CPU, memória e disco. Sua arquitetura consiste em substituir as chamadas MPI pela aplicação Colza, que possui funcionalidades de gerenciamento de processos, comunicação e controle de elasticidade. Sua solução foi avaliada de forma a garantir com que a visualização gerada pelo software Paraview se mantivesse constante a medida que a malha resultante da simulação se tornasse mais complexa a cada iteração.

Nota-se dos trabalhos acima mencionados, o foco em pontos como tratamento de falhas, minimizar o tempo de provisionamento de instâncias, técnicas de monitoramento de recursos e automatização no gerenciamento da elasticidade, etc, com soluções que vão desde a camada da infraestrutura (NAT) a novas implementações da biblioteca MPI. Já a solução aqui proposta oferece as funções da interface MPI e algumas adicionais para provisionamento de instâncias em nuvem, adição e remoção de nós de processamento visando prover elasticidade. Permite montar uma plataforma única de execução com servidores e *clusters* localizados em redes distintas. Aplicações com chamadas de apenas funções da interface MPI não necessitam de recompilação. O procedimento para execução direciona as chamadas de funções para a biblioteca MPI nativa ou MCMPI.

3. Apresentação da Biblioteca MCMPI

Esta seção apresenta a biblioteca MCMPI que estende a interface MPI. Seu objetivo é voltado para a formação de uma plataforma de execução que agrega servidores e *clusters*. A extensão provê funções de elasticidade do ambiente de execução quanto aos recursos de processamento.

A seguir são apresentados a interface da biblioteca MCMPI, um exemplo de programa de usuário e a descrição da arquitetura.

3.1. Interface da Biblioteca MCMPI

A programação de uma aplicação usando a interface MPI efetua chamadas de funções da biblioteca MCMPI. São aproveitadas as funções nativas da distribuição MPI não o restringindo a nenhuma versão específica. É aplicado o uso do mecanismo de LD_PRELOAD do Linux, que possibilita controlar uma função chamada pela aplicação para executar a implementação MPI nativa ou MCMPI. Dessa forma, garante-se a execução da aplicação de forma transparente sem a necessidade de recompilar a aplicação.

Na versão atual da biblioteca MCMPI, foram implementadas as funções de inicialização, finalização, sincronização e comunicação da interface MPI. Além de funções adicionais para provisionamento de servidores na *Cloud*, adição e remoção de nós, conforme lista a seguir:

- `MPI_Init`: Função inicia com a execução da MPI_Init nativa e em seguida realiza todo o processo de leitura do arquivo de configuração e criação dos processos, montando a plataforma de execução do usuário.
- `MPI_Send`: Correspondente à MPI_Send nativa, provendo a conversão do *rank* do processo destino e o envio.
- `MPI_Recv`: Correspondente à MPI_Recv nativa, provendo a conversão do *rank* do processo destino e o recebimento.
- `MPI_Comm_Rank`: Retorna o *rank* do processo. Apenas o comunicador MPI_COMM_WORLD é disponibilizado.
- `MPI_Comm_Size`: Retorna o número total de nós ativos.
- `MPI_Barrier`: Função de sincronização que garante que todos os *ranks* estão naquele trecho antes de seguir adiante.
- `MCMPI_Provision_Cloud_Server`: Função que provisiona uma instância na nuvem sob demanda. Na biblioteca MCMPI corrente apenas o provisionamento EC2 na nuvem AWS está disponível.
- `MCMPI_Add_Node`: Função que adiciona um novo *rank* à aplicação em tempo de execução.
- `MCMPI_Remove_Rank`: Função que remove um *rank* existente em tempo de execução.
- `MCMPI_Add_Cluster`: Função que adiciona um *cluster* inteiro à plataforma de execução.
- `MCMPI_Remove_Cluster`: Função que remove o *cluster* inteiro especificado.

Uma execução MPI consiste na execução de um ou mais processos MPI em um ou mais servidores. Cada processo, identificado por meio de *ranks*, é passado para o usuário na forma de um inteiro iniciando de 0 a $N - 1$.

3.2. Exemplo de execução

A biblioteca MCMPI está voltada para atender 2 tipos de aplicações: aquelas que necessitam de uma plataforma de execução com múltiplos domínios, incluindo servidores, *clusters* e recursos computacionais em nuvem, que são agregados no início da execução e outras com perfil de elasticidade, em que tais recursos são incorporados dinamicamente durante a execução.

No primeiro tipo de aplicações, o usuário precisa criar uma variável de ambiente de chave “MCMPI_HOSTFILE” apontando o caminho do arquivo de configuração como valor como descreve a Figura 2. O conteúdo do arquivo de configuração lista os servidores e *clusters* a serem utilizados naquela execução como mostra a Figura 3. O comando `mcmpirun` é somente um *script bash* que adiciona o LD_PRELOAD ao `mpirun`.

```
1 export MCMPI_HOSTFILE="/caminho/arquivo/mcmpi_hostfile"
2 mcmpirun ./programa
```

Figura 2: Exemplo de configuração do *hostfile* para infraestrutura apresentada.

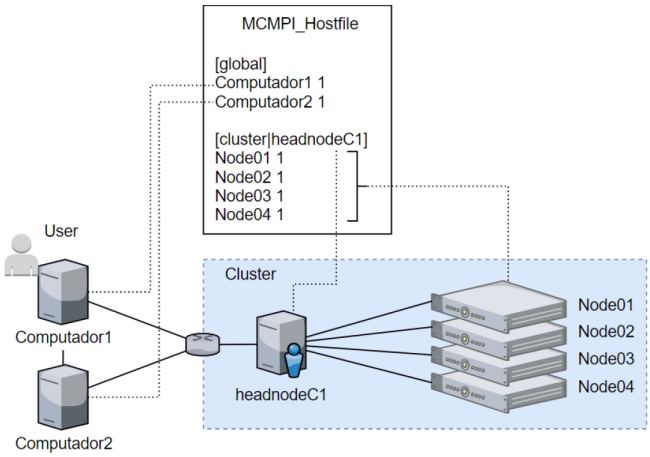


Figura 3: Exemplo de configuração do *hostfile* para infraestrutura apresentada.

No segundo tipo de aplicações, a biblioteca MCMPI oferece ao usuário funções para provisionamento de novas instâncias na *cloud*, como também sua adição e remoção conforme mostra a Figura 4.

```
1  #include <mpi.h>
2  #include <mcmpi.h>
3  int main(int argc, char ** argv){
4      MPI_Init(NULL, NULL); // Inicia plataforma de execução
5      ... // Executa aplicação do usuário
6      MCMPI_Add_node("hostname1"); // Adiciona novo processo MPI no nó <hostname1>
7      // Provisiona novo nó na cloud com retorno de hostname e/ou IP
8      char * new_hostname = MCMPI_Provision_Cloud_Server("<AWS_Profile>", 1, "<instance_flavor>");
9      MCMPI_Add_node(new_hostname); // Adiciona host recem criado
10     ... // Executa aplicação do usuário
11     MCMPI_Remove_Rank(1); // Remove rank de ID = 1
12     ... // Executa aplicação do usuário
13 }
```

Figura 4: Exemplo de configuração do *hostfile* para infraestrutura apresentada.

A Figura 4 ilustra o exemplo de uma aplicação com perfil de elasticidade . Na linha 6 é adicionado um servidor já existente de *hostname* “hostname1”. Em seguida, é provisionado um novo servidor na nuvem e seu IP ou *hostname* armazenado na variável de retorno “new_hostname”, que é adicionado à aplicação através do comando `MCMPI_Add_Node` na linha 9. Na linha 11 o processo de *rank* 1 é removido.

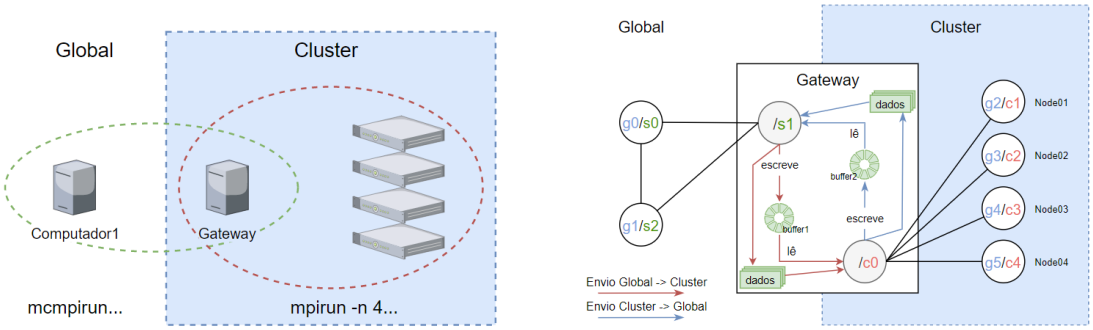
3.3. Descrição da Arquitetura

O principal desafio de se adicionar um *cluster* e/ou recursos da nuvem para execução de uma aplicação MPI é a sua localização em domínios diferentes. Por conta de segurança, esses recursos se encontram em redes isoladas atrás de *firewalls* onde normalmente é configurado um servidor que intermediará o acesso a estes servidores. Este servidor é comumente denominado *Bastion Host* na nuvem e Nó de Login ou *headnode* em *clusters*.

A solução escolhida para abordar tal problema foi executar o comando do `mpirun` de forma isolada em cada domínio e depois integrá-los numa visão única a ser entregue para a aplicação do usuário como mostra a Figura 5a, que consiste num comunicador global (representado pelo grupo tracejado em verde) e o comunicador do *cluster*

(grupo vermelho). A biblioteca MCMPI gerencia uma tabela chamada "Tabela de Controle" com o objetivo de mapear a localidade de cada processo MPI fazendo com que a troca de mensagens entre os processos seja feita de forma transparente para o usuário. O nó *gateway* é o único a fazer parte dos dois comunicadores, sendo, portanto peça chave para garantir a comunicação entre os dois comunicadores, como apresentado na Figura 5b.

O "Gateway" é responsável pela comunicação dos nós internos do *cluster* com os nós externos. O *buffer1* e o *buffer2* possuem a estrutura em anel de forma a garantir o envio das mensagens do comunicador interno para o externo e vice-versa . Os *buffers* carregam somente os metadados da mensagem. Os dados da mensagem são carregados num *buffer* secundário e apagados depois do envio ao processo destino.



(a) Comunicadores numa execução MCMPI

(b) Processo interno do *gateway* para troca de mensagens.

Figura 5: Comunicadores e *Gateway* na MCMPI

A Tabela 1 ilustra como seria preenchida a Tabela de Controle para a plataforma representada na Figura 3. Nela, a coluna de nome *global_rank* lista os processos com os respectivos *ranks* a serem entregues para a aplicação do usuário. A coluna *local_rank* lista os *ranks* locais para cada grupo de comunicador contido na coluna *parent_comm*. Foram colocados caracteres prefixo para facilitar a identificação de cada *rank*, sendo: "s" (s0, s1) comunicadores do tipo sistema, "c" (c0,c1) identificados pelo *cluster* e "g" (g0,g1) *rank* do tipo global e de visualização do usuário. Na implementação da MCMPI, tais campos referentes a *ranks* são números inteiros (0, 1, 2, 3...).

nodename	local_rank	global_rank	parent_comm
Computador1	s0	g0	s0
Computador2	s2	g1	s0
Gateway	s1	NULL	s1
Gateway	c0	NULL	NULL
Node01	c1	g2	s1
Node02	c2	g3	s1
Node03	c3	g4	s1
Node04	c4	g5	s1

Tabela 1: Visão da Tabela de Controle da plataforma da Figura 3.

A Tabela de Controle possui todas as informações para que um processo com determinado *rank* envie a mensagem para seu destinatário corretamente. Tomando a execução da Figura 3, Figura 5b e Tabela 1 como exemplo, tem-se que existe conexão direta entre “g0” e “g1”, mas o envio de uma mensagem de “g0” a “g5” precisará ser roteada pelo *gateway* passando pelos processos “s1’ e “c0”. Tudo isso é feito pelo mecanismo do LD_PRELOAD, que sobrescreve a função de comunicação (e.g. `MPI_Send`) pela respectiva da MCMPI, que direciona as mensagens como mostra a Figura 6b.

```
Se rank 0:
    Lê arquivo MCMPI_HOSTFILE
        Se o recurso for servidor:
            MCMPI_Add_Node(...);
        Se o recurso for cluster onpremise:
            MCMPI_Add_Cluster(...);
        Se o recurso for instância da nuvem:
            MCMPI_Provision_AWS_Server(..., ConfigA);
            MCMPI_Add_Node(...);
        Se o recurso for cluster na nuvem:
            Para cada instancia do cluster:
                MCMPI_Provision_AWS_Server(..., ConfigB);
            MCMPI_Add_Cluster(...);
Senão:
    Aplicação do usuário
```

(a) MPI_Init da MCMPI

```
MPI_Send(mensagem, destino){
  Inicio
    Verificar informações de nós origem e destino na tabela
    de controle
    caso 1:
      Se origem.parent_node = destino.parent_node
      Então MPI_Send(mensagem, destino)
    caso 2:
      Se origem.parent_node != destino.parent_node
      nova_mensagem = mensagem + ação + nó destino
      MPI_Send(nova_mensagem, destino.parent_node)

  /**
  Onde o caso 1 seria o caso normal onde os nós origem
  e destino compartilham o mesmo comunicador local.
  E o caso 2 onde os nós origem e destino não compartilham
  o mesmo comunicador cabendo ao remetende enviar para o
  parent_node do destinatário.
  */
}
```

(b) MPI_Send da MCMPI

Figura 6: Pseudocódigos para funções MPI_Init e MPI_Send da MCMPI.

O processo de *rank* 0 é o primeiro a ser inicializado numa execução MCMPI. Após inicializado, executa a função MPI_Init da biblioteca MCMPI, criando a plataforma de execução especificada pelo usuário no arquivo MCMPI_HOSTFILE como apresenta a Figura 6a.

`MCMPI_Add_Node` é uma das funções oferecidas pela biblioteca MCMPI. Ela provê a criação de um processo MPI num servidor e o adiciona à aplicação corrente. Como descreve a Figura 7, o *controller* (*rank* 0) inicia um novo processo no servidor “Server02” por meio do comando `mpirun` num cenário onde já existe o “Server01” criado. O *controller* primeiro notifica os demais *ranks* que um novo processo está sendo adicionado. O processo no Server01 é iniciado chamando a função `MPI_Init` da MCMPI que em seguida inicia o `MPI_Init` nativo. Os passos seguintes envolvem o envio do *hostname* e *rank* local pelo processo adicionado e recebimento do *rank* global. Nota-se, também, a inicialização de uma *thread* “thread_work”, que permanece em plano de fundo. A *thread_work* desempenha um papel fundamental na adição de novos nós uma vez que é utilizada na integração do novo *rank* ao comunicador atual.

Como apresentado na Figura 5a, o processo de adição de um *cluster* pela função `MCMPI_Add_Cluster` envolve a criação do ambiente de execução no *cluster* e depois, sua integração lógica à aplicação do usuário por meio da Tabela de Controle. O processo se inicia da mesma forma que o `MCMPI_Add_Node`, exceto que, no envio do *rank* global, é enviado um inteiro especial iniciando a execução da aplicação MPI no nó *Gateway* do *cluster*, criando dois *buffers* para a comunicação de mensagens, conforme mostrado na Figura 5b. Iniciada a execução da aplicação nos nós do *cluster*, os *ranks* são coletados

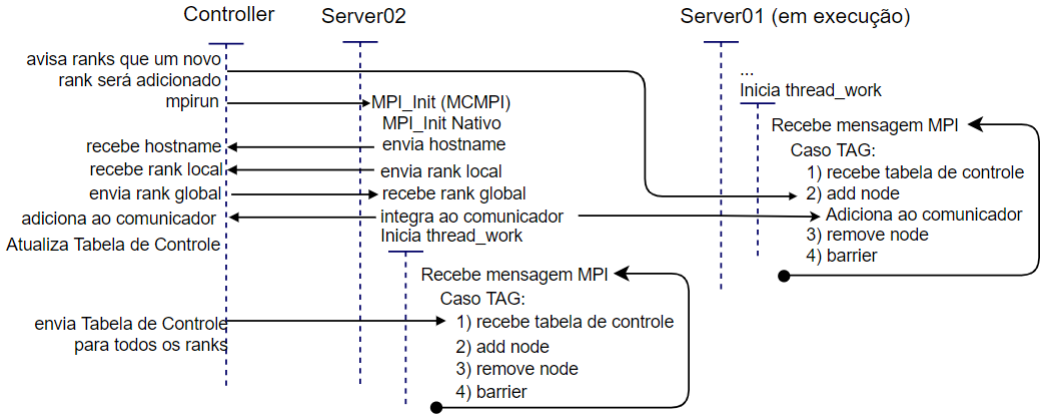


Figura 7: Diagrama de Sequência da Função `MCMPI_Add_Node` no Server02

e associados aos respectivos *ranks* globais. Tal informação é consolidada na Tabela de Controle e depois compartilhada com os demais *ranks*.

A biblioteca MCMPI oferece a possibilidade de se provisionar nós na nuvem. Na atual versão, somente o provedor AWS é suportado, mas não limita-se a ela. A solução foi desenvolvida de forma que a comunidade pudesse criar seus próprios *drivers* e adaptadores aos mais diversos tipos de orquestradores e provedores.

Para a nuvem é necessário as "Configs", que é onde se mapeia a configuração com a qual será criada a instância. Nelas o usuário poderá configurar propriedades como número de vCPUs, memória RAM, quantidade e tamanho dos discos, *subnet*, etc. Na nuvem é possível escolher entre dois tipos de *subnet*: privada e pública. Na *subnet* pública todos os servidores ficam expostos na Internet enquanto que na privada estes são isolados. A preferência de escolha depende do grau de segurança da organização e do *sysadmin* que a gerencia. Ressalta-se que a biblioteca suporta ambas as configurações com a particularidade de que, para adição de *clusters* localizados em *subnets* privados, é necessário utilizar a inclusão no modo *cluster* (`MCMPI_Add_Cluster`) apontando a instância *Bastion* como *gateway*.

A biblioteca MCMPI também oferece a remoção de recursos computacionais. Tais recursos podem ser adicionados e removidos de forma dinâmica durante a execução da aplicação do usuário como apresentado nas seções anteriores.

4. Resultados

Foi implementado um protótipo da biblioteca MCMPI utilizando como base a versão OpenMPI 4.0.1. Como forma de demonstrar as funcionalidades, consistência e desempenho da aplicação, foram realizados quatro experimentos, a serem detalhados com maior profundidade ao longo desta seção. O "Experimento 1" teve a finalidade de comparar o desempenho da MCMPI frente ao MPI nativo. Os experimentos "Experimento 2" e "Experimento 3" avaliaram o suporte e o desempenho da biblioteca MCMPI para execução de aplicações em múltiplos domínios. E, por fim, o "Experimento 4" analisou os recursos para se criar aplicações elásticas com possibilidade de adição e remoção de nós de forma dinâmica.

Todos os testes foram realizados na plataforma AWS (*Amazon Web Services*) onde

foram utilizadas instâncias EC2 (*Elastic Compute Cloud*) do tipo `m5a.xlarge`, que consiste em uma máquina com processador AMD EPYC 7571 com 4 vCPUs e 16GB de RAM. Durante os testes foram alocados um ou mais processos MPI para cada instância. O sistema operacional instalado nas instâncias é o Amazon Linux 2 AMI (*Amazon Machine Image*) com kernel 5.10.

No "Experimento 1", tem-se o objetivo de analisar o comportamento das funções de comunicação como também avaliar o custo da comunicação em relação ao MPI Nativo. Foram utilizados dois *workloads*: o código de multiplicação de matrizes e uma aplicação que simula a distribuição de calor num mapa 2D (HeatMap 2D). Ambas as aplicações são disponibilizadas pela Lawrence Livermore National Laboratory (LLNL) [LLNL Matrix Multiply].

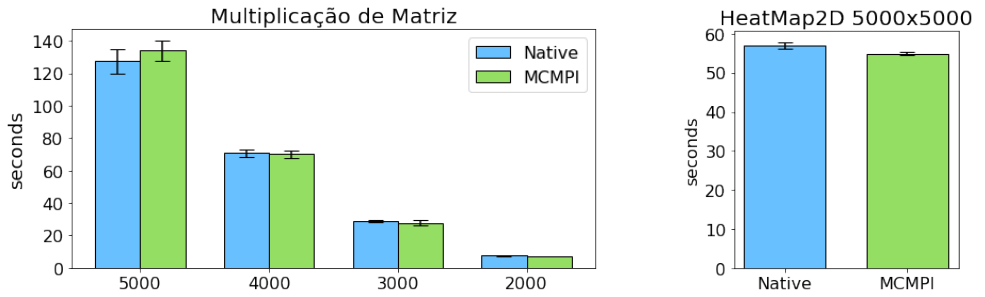
Foram escolhidas essas duas aplicações por representarem cenários de comunicação distintos: a multiplicação de matrizes possui um padrão de envio de uma pequena quantidade de mensagens de tamanhos grandes e a aplicação HeatMap 2D realiza uma grande quantidade de troca de mensagens de tamanhos pequenos.

Para ambos os testes foram utilizados 12 processos MPI a serem alocados em 12 instâncias EC2. Para os testes da multiplicação de matrizes utilizou-se blocos (NB) de tamanho 5000×5000, 4000×4000, 3000×3000 e 2000×2000. E para o *HeatMap 2D* utilizou-se um bloco de 5000×5000 com 1000 iterações.

Neste experimento, foram realizadas dez execuções para cada aplicação. Os tempos de execução obtidos foram analisados estatisticamente com o tratamento de *outliers* e o cálculo de média aritmética, desvio padrão e intervalo de confiança de 95%.

A Figura 8 mostra que as diferenças de desempenho foram mínimas quando comparado as execuções dos dois *workloads* com MCMPI e o MPI nativo. A MCMPI obteve melhor tempo de execução em quase todos os testes realizados com um desempenho superior de até 9%. O MPI nativo obteve um tempo 4,8% menor para a multiplicação de matrizes 5000×5000, mantendo essa diferença com matrizes de 6000×6000 e 7000×7000 em menos de 2,5%.

Os Experimentos 2 e 3 tem como objetivo analisar e avaliar a funcionalidade e o comportamento das funções de adição de *clusters* na plataforma de execução e suas comunicações. Foram executadas as mesmas aplicações apresentadas no "Experimento

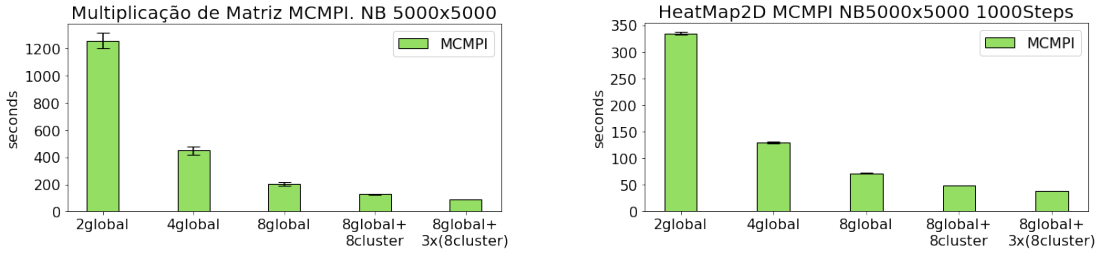


(a) Multiplicação de Matrizes (NB Variado) **(b)** HeatMap 2D

Figura 8: Quadro comparativo MPI Nativo vs MCMPI

1”: multiplicação de matrizes e dissipação de calor 2D. Nestes experimentos objetivou-se demonstrar a execução das aplicações agregando *clusters* localizados em domínios distintos, ao invés de usar somente servidores encontrados na mesma rede da máquina principal.

Ressalta-se que o uso de nós em domínios distintos, como em um ambiente incluindo *clusters*, não é uma funcionalidade existente na implementação nativa do MPI, não sendo possível, portanto, uma comparação direta entre eles. Nos experimentos descritos a seguir, foram realizados testes com 2 *ranks*, 4 *ranks*, 8 *ranks*, 16 *ranks* (sendo 8 locais e 8 de um *cluster*) e 32 *ranks* (sendo 8 locais mais 3 *clusters* contendo 8 *ranks* cada).



(a) **(b)**

Figura 9: Desempenho da MCMPI para inclusão de *clusters* de domínios diferentes

A partir dos resultados apresentados na Figura 9, percebe-se que ambas as aplicações se beneficiam com uma diminuição no tempo de execução com o aumento do número de nós.

Destaca-se, também, a importância do papel do *gateway*, principalmente na execução da multiplicação de matrizes. Por este transmitir uma grande quantidade de dados. É interessante apontar um cenário onde, no pior caso, todos os processos de dentro de um domínio enviam dados para um processo de fora, ficando estes em espera nos *buffers* no *gateway* do *cluster* de origem. É essencial que o desenvolvedor da aplicação deve se atentar para que a soma das mensagens em trânsito não ultrapassem o tamanho máximo da memória disponível daquela máquina.

No “Experimento 4” são apresentadas duas funcionalidades exclusivas da biblioteca MCMPI com o objetivo de oferecer elasticidade à aplicação. São elas a capacidade de se adicionar e remover nós de forma dinâmica no decorrer da execução da aplicação, através das chamadas das respectivas funções de adição e remoção de nós. A fim de demonstrar tais funcionalidades, foi criada uma aplicação, conforme apresentado na Figura 10a, que simula uma aplicação cliente-servidor intermediados por um *gateway* cujo papel é orquestrar as requisições dos clientes para os servidores (*workers*). O *gateway* distribui as requisições no modo produtor-consumidor de forma que, caso não haja *worker* disponível, este é então armazenado numa fila. O *gateway* então é capaz de decidir se precisa adicionar ou remover nós a partir do monitoramento do tamanho da fila.

Foi realizado um teste com 800 requisições, sendo as 400 primeiras a uma taxa de 1,8rps (requisições por segundo), seguidas de 50 a 2,5rps e as últimas 350 a 2rps,

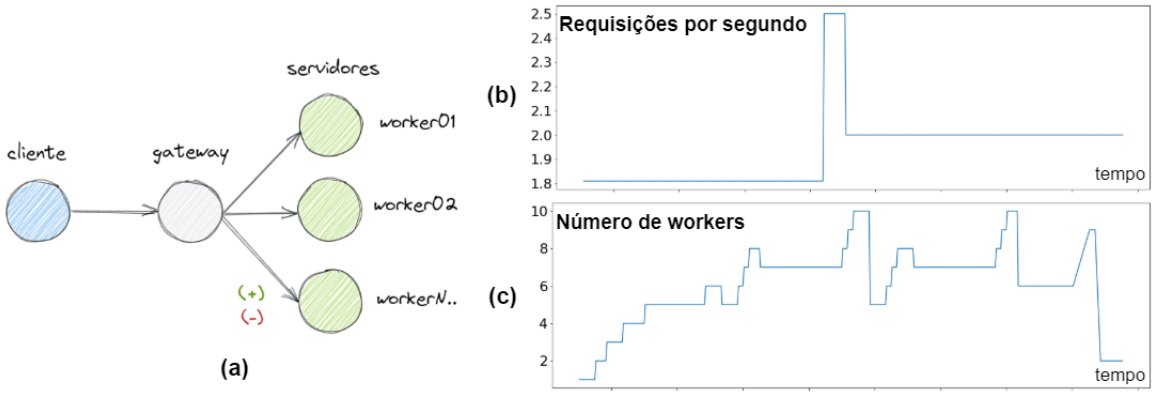


Figura 10: Arquitetura da aplicação do Experimento 4.

conforme mostra a variação da vazão de requisições no gráfico da Figura 10b, a fim de se ter um cenário em que são necessárias adições e remoções de nós para manter o tempo de resposta dentro de uma faixa de valores. Foi configurado, também, o tempo fixo de 3 segundos para os *workers* processarem as requisições.

A heurística utilizada para a decisão de remoção e adição de nós é a mesma utilizada no modo *TargetValue* do projeto open source [Kubernetes], sendo:

$$desiredReplicas = \left\lceil currentReplicas \times \left(\frac{currentMetricValue}{desiredMetricValue} \right) \right\rceil$$

onde foi adotado um valor desejável da métrica, aqui sendo o tamanho da fila, o número de réplicas (*workers*) se ajustará automaticamente referente ao valor corrente da métrica. Por exemplo: ao estipularmos um valor *desiredMetricValue* de 10, ou seja, 10 requisições para cada *worker*. Se em dado momento o tamanho da fila é de 20 requisições e o número de *workers* é 1, o cálculo de *desiredReplica* $(1 \times (\frac{20}{10})) = 2$ indicaria que precisaríamos adicionar um novo *worker* totalizando dois *workers* no *pool*.

Para a realização do teste de desempenho, foram utilizados tamanhos maiores de instâncias EC2 do tipo `c6a.2xlarge` com 8 vCPU e 16GB RAM e, para cada servidor foram alocados um número máximo de 6 processos MPI. Somente quando saturado o número de processos MPI em todos os servidores é realizado o provisionamento de uma nova instância. Provisionamento esse que leva de 2 a 3 minutos, o que é possível constatar na Figura 10c pela ocorrência de um período constante do número de *workers* seguido por um incremento.

O gráfico apresentado na Figura 11 mostra que o tempo de resposta foi mantido dentro do intervalo entre 30 e 60 segundos, conforme o objetivo especificado.

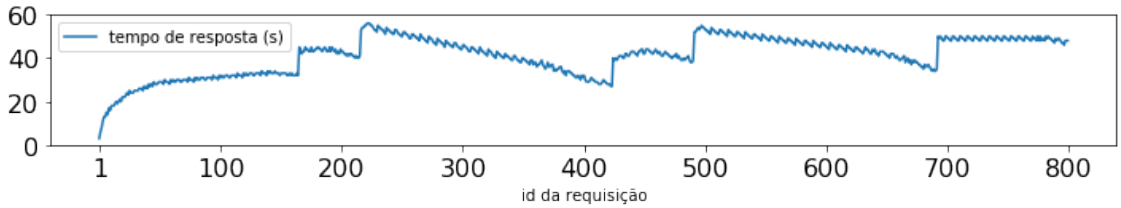


Figura 11: Duração em segundos de cada requisição.

5. Considerações Finais e Trabalhos Futuros

A biblioteca MCMPI mostrou-se capaz de entregar uma plataforma de execução que agrega servidores, *clusters* e recursos da nuvem localizados em múltiplos domínios de forma transparente e com desempenho. O usuário apenas define um arquivo de configuração e executa a aplicação sem a necessidade de recompilar o código. Foi apresentado, também, um protótipo de solução elástico que utiliza as funções de provisionamento de instâncias na nuvem, adição e remoção de nós em tempo de execução. Tal implementação foi feita de forma flexível de modo a facilitar o suporte aos demais provedores de nuvens (e.g. Azure, Google) e orquestradores (e.g. Kubernetes, YARN). Como trabalhos futuros volta-se a atenção no suporte aos demais provedores de nuvem, à implementação de mais funções MPI coletivas (`MPI_Gather`, `MPI_Scatter`, etc).

Referências

[A. Raveendran 2011] A. Raveendran, T. Bicer, G. A. (2011). A framework for elastic execution of existing mpi programs. *IEEE International Conference on Parallel and Distributed Processing Symposium*, pp.940-047.

[Choi et al. 2004] Choi, S., Park, S., Han, S., Park, S., Kwon, O., Kim, Y., and Park, H. (2004). An nat-based communication relay scheme for private-ip-enabled mpi over grid environments. *Computational Science - ICCS 2004*, Volume 3036 of the series Lecture Notes in Computer Science:pp. 499–502.

[Dorier et al. 2022] Dorier, M., Wang, Z., Ayachit, U., Snyder, S., Ross, R., and Parashar, M. (2022). Colza: Enabling elastic in situ visualization for high-performance computing simulations. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 538–548.

[Kubernetes] Kubernetes. `https://kubernetes.io/`. Acesso em 21/07/2022.

[LLNL Matrix Multiply] LLNL Matrix Multiply. Llnl matrix multiply. `https://hpc-tutorials.llnl.gov/mpi/examples/mpi_mm.c`. Acesso em 19/07/2022.

[M. Caballer 2021] M. Caballer, M, A. Z. M. P. G. M. (2021). Deployment of elastic virtual hybrid clusters across cloud sites. *Journal of Grid Computing*, vol. 19, n. 4, pp. 1-16.

[Massetto 2007] Massetto, F. I. (2007). Hybrid mpi - uma implementação mpi para ambientes distribuídos híbridos. *Tese de doutorado. Escola Politécnica da USP.*

[P. Patchin 2009] P. Patchin, H. A. Lagar-Cavilla, E. d. L. M. B. (2009). Adding the easy button to the cloud with snowflock and mpi. *Association for Computing Machinery*.