Desafios e Abordagens para Alocação Dinâmica de Recursos em Sistemas HPC com MPI

Paulina Rehbein[1], **Dalvan Griebler**[1]

[1]Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

paulina.ester@edu.pucrs.br, dalvan.griebler@pucrs.br

Resumo. *Este artigo descreve brevemente o problema da alocação dinâmica de recursos em ambientes HPC com MPI, discutindo tipos de alocação e escalonamento, funcionalidades das versões do MPI e abordagens para mudança de recursos em aplicações MPI. Ao final, propõe uma revisão da literatura e a avaliação das principais ferramentas de alocação de recursos em clusters HPC.*

1. Introdução

A crescente demanda, da área cientifica e principalmente industrial, por aplicações MPI (*Message Passing Interface*) adaptáveis tem impulsionado o interesse por técnicas que permitam ajustar dinamicamente o uso de recursos durante a execução de aplicações. Nesse contexto, surge o conceito de maleabilidade, que se refere à capacidade de uma aplicação alterar, em tempo de execução, a quantidade de recursos computacionais como número de processos, e nós computacionais, sem necessidade de reinicialização. Diferentemente de aplicações estáticas, que operam com uma alocação fixa de recursos do início ao fim, aplicações maleáveis conseguem expandir ou reduzir sua escala de execução conforme a disponibilidade de recursos ou as necessidades da própria aplicação. A maleabilidade implica desafios como a redistribuição de carga, reorganização de estruturas de comunicação e manutenção da consistência do estado, mas em contra partida trás benefícios como o melhor uso de recursos computacionais, possibilitando distribuição dinâmica de recursos, ajustando-os às cargas das aplicações em execução.

A elasticidade ou maleabilidade de processos foi amplamente estudada e desenvolvida para aplicações comerciais de serviços hospedados na internet e executados em provedores de nuvem comerciais como AWS, Azure e Google Cloud. Entretanto, o mesmo ainda não aconteceu nos ambientes de HPC (computação de alto desempenho) que utilizam MPI devido à combinação de 3 fatores: (1) baixa demanda por essa funcionalidade durante a época da concepção do modelo MPI em meados dos anos 90; (2) necessidade de padronização da metodologia, o que é um processo lento uma vez que envolve consenso de universidades, laboratórios e grandes empresas do mundo todo. Discussões relacionadas já acontecem há cerca uma década no comitê do MPI; (3) gigantesca complexidade de implementação devido à necessidade de suporte de toda a pilha de software de HPC, incluindo provedor de recursos (p.e., Slurm, Torque ou Flux), *runtime* (p.e., Hydra ou PRRTE), manejador de recursos (p.e., PMIx ou PMI), e API de programação (p.e., OpenMPI e MPICH), e mudanças nas aplicações em si, totalizando milhões de linhas de código.

2. Fundamentação Teórica

Com o objetivo de explicar os conceitos básicos e entender a complexidade da alocação de recursos em aplicações HPC utilizando MPI, a seguir serão descritos os tipos de alocação e agendamento de *jobs* paralelos existentes, as versões do MPI que incluiram funcionalidades relacionadas a execução de *jobs* paralelos e mudança de recursos, e quais são as estratégias de adaptação de recursos no contexto do HPC que são conhecidas.

2.1. Alocação e agendamento de *parallel jobs*

Diferentes conceitos sobre alocações e agendamento de *jobs*, no contexto de aplicações paralelas, *parallel jobs*; estes são compostos de atividades de comunicação independentes e, segundo [Feitelson and Rudolph 1996] et al., podem ser divididos em quatro tipos baseando-se no número de processadores que serão utilizados pelo *job*: **(1) Rigid Jobs** precisam de um número definido de processadores para executar. Um *Rigid Job* não será executado com uma quantidade menor de processadores e não irá adicionar mais processadores durante na execução; **(2) Evolving Jobs** são *jobs* que podem ter seus requisitos de recursos alterados a pedido da aplicação, se o sistema não puder satisfazer esses recursos, então a execução da aplicação não poderá continuar; **(3) Moldable jobs** pode ter um número flexível de processadores, e precisa que o sistema aloque um número de processadores dedicados. O número de processadores deve ser definido no início da execução, e o *job* irá adaptar esse número de processadores. Depois que a execução iniciar, não será mais possível reconfigurar o *job*; e **(4) Malleable jobs** é o tipo mais flexível de *job*, podendo se adaptar a mudanças no número de processadores durante a execução. Podendo ser um objeto maleável, que é aquele que pode sofrer mudanças no formato por ação de uma entidade externa, ou um objeto em evolução, que muda por conta própria. Aplicações desse tipo podem ter todos os seus processadores retirados para execução de outras atividades, mantendo a execução parada e iniciando novamente quando houver recursos disponíveis.

2.2. *Message Passing Interface*

O padrão MPI (*Message Passing Interface*) descreve um conjunto base de rotinas para programação de sistemas de computadores distribuídos conectados por uma rede de comunicação. Proposto no início dos anos 90, em um esforço conjunto de laboratórios, universidades e grandes empresas, o MPI surgiu com o intuito de padronizar, de forma clara, o paradigma de programação que antes era mantido de forma independente por diversos fabricantes e laboratórios, como IBM, Intel, Cray e Oak Ridge. A cada versão do MPI surgiam novas funcionalidades priorizando as necessidades da comunidade científica e industrial da época, as relevantes para esse trabalho são: **MPI 1.0** (1994) estabeleceu a base das rotinas necessárias para computação distribuída; **MPI 2.0** (1997) como descrito em [MPI Forum 1997] definiu a comunicação unilateral, I/O paralelo e criação de novos processos (fixo à alocação inicial do *job*); **MPI 4.0** (2021) [MPI Forum 2026] introduziu do modelo de Seções MPI, comunicação particionada e otimizações diversas de desempenho e escalabilidade; **MPI 5.0** (2025-presente) fez a introdução da ABI, refinamento do modelo de Seções, e exitem discussões para definir maleabilidade/elasticidade de recursos. Vale ressaltar que, nas versões anteriores à 4.0, como apontado por [Streubel 2020] et al. só é possível inicializar a *runtime* do MPI uma única vez (`MPI_Init`), e todos os processos precisam ser conhecidos durante a inicialização, para que o comunicador global, que é imutável, seja criado. E quando um dos processos é finalizado (`MPI_Finalize`), ele torna o comunicador global inválido, terminando com a execução da aplicação.

2.3. Estratégias de Alocação Dinamica de Recursos

Existem diferentes estratégias que possibilitam a adição/remoção dinâmica de processos MPI, na literatura foram encontradas duas principais:

(1) ***Checkpoint/Restart***

Onde a aplicação deve guardar o estado de execução da aplicação em determinados intervalos de tempo, o estado armazenado deve ser adequado o suficiente para aplicação

poder iniciar e executar a partir dele, chamado de *checkpoint*. Assim, quando uma ação de remover ou adicionar processos é executada, um *checkpoint* é registrado, a aplicação é finalizada e iniciada com o novo número de processos. Metodologias para tolerância a falhas, balanceamento do *workload* da aplicação e ferramentas de monitoramento podem ser incorporadas, de modo que possibilite a mudança de processos de forma automática, com o objetivo de ter a melhor combinação de recursos em relação a cada aplicação que está em execução. O ônus dessa abordagem é a diminuição do desempenho pela necessidade de salvar o estado, finalizar a aplicação, alocar novos recursos, iniciar a aplicação, recuperar o estado salvo e realizar o balanceamento de carga aos recursos existentes.

(2) ***MPI Sessions***

É uma solução atual, tendo em vista que o conceito de *Sessions* no MPI foi introduzida na versão 4.0, trazendo novos conceitos como as sessões, *PSets* (*Process Sets*), e reutilizando outros já bem definidos como os grupos e comunicadores. Ao definir o *MPI Sessions*, as rotinas `MPI_Init` e `MPI_Finalize` não são necessárias, e várias sessões podem ser criadas durante a execução da aplicação com a função `MPI_Session_init`. Os processos que são inicializados juntos fazem parte da mesma sessão MPI e do mesmo *PSet* (`mpi://WORLD`). Outros *PSets* podem ser definidos durante a execução e a partir deste é possível criar um grupo e então um novo comunicador. Para destruir uma sessão MPI especifica, basta chamar a função `MPI_Session_finalize`, mantendo as sessões remanescentes e futuras válidas. Quando a dinâmica de sessões, *Psets*, grupos e comunicadores for bem definida e implementada nas APIs de programação (OpenMPI e MPICH) nas suas futuras versões, irá permitir a alocação dinâmica de processos, e adicionar um novo nível de complexidade ao programador, sendo necessário garantir a consistência da comunicação, salvar o estado dos processos que serão removidos e fazer o balanceamento de carga aos que existem e/ou que serão adicionados.

Dessa forma, podemos relacionar os tipos de *jobs* citados em 2.1, as versões MPI apontadas em 2.2 e as estratégias de alocação citadas acima, e podemos observar que: o **MPI 1.0** faz parte dos *Rigid jobs*, uma vez que não é possível utilizar um número diferentes de processos do que foi definidos ao executar, e só é possível realocar recursos por meio de *checkpoint/restart*; **MPI 2.0** sendo mais próximo de *Moldable jobs*, onde os processos que foram fixamente alocados no *job*, podem ser adicionados na aplicação durante a execução, como nessa versão a remoção de processos não é suportada e a quantidade de recursos deve ser fixa no *job*, a remoção completa de recursos alocados só pode ser feita por *checkpoint/restart*; e o início do *Malleable jobs*, o estado da arte da alocação dinâmica de recursos, nas versões **MPI 4.0** e **MPI 5.0**, que ainda estão em evolução. Além disso a mudança de recursos por *checkpoint/restart* pode ser classificada como *Evolving Jobs*, pois só é possível continuar (iniciar novamente) a execução da aplicação se os recursos solicitados estão disponíveis.

3. Proposta

Considerando que não existe uma solução de maleabilidade de recursos MPI bem definida, consolidada, padronizada e amplamente utilizada em ambientes de produção HPC, será conduzida uma revisão da literatura com o objetivo de identificar abordagens que proponham mecanismos de expansão e redução dinâmica de recursos em aplicações baseadas em MPI a fim de identificar suas limitações técnicas e avaliar seu grau de maturidade quanto à adoção em ambientes reais de produção. A metodologia para a revisão da literatura envolverá uma busca por artigos e periódicos em 3 bases de dados científicas: Scopus, WebOfScience e IEEEXplore. O texto de busca demonstrado na Figura 1, com

apenas pequenas alterações de sintaxe para as demais bases de dados. Essa busca contempla diferentes palavras-chave em inglês, como maleabilidade, elasticidade e dinâmico, ao mesmo tempo que exclui palavras-chave relacionadas à inteligência artificial e ao aprendizado de máquina em geral. Há também uma delimitação que busca apenas retornos de trabalhos pertinentes à subárea de Ciência da Computação. A partir desse mapeamento, será realizada uma análise comparativa das principais abordagens encontradas, considerando aspectos como estratégia de maleabilidade, complexidade de implementação, impacto no desempenho, tolerância a falhas, distribuição MPI utilizada e integração com gerenciadores de recursos utilizados em ambientes HPC, como o SLURM. Essa etapa permitirá identificar problemas técnicos e limitações recorrentes que dificultam a adoção prática dessas soluções. Além disso, serão definidos cenários experimentais para validação das

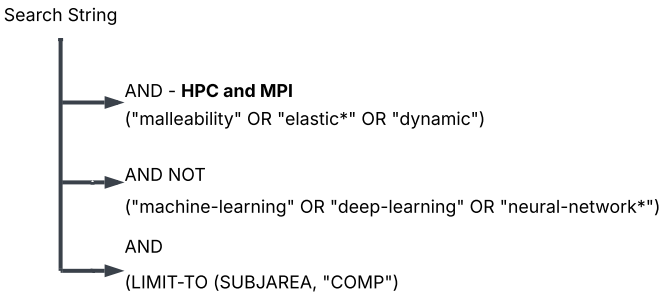

Figura 1. *String* de busca decomposta nos operadores lógicos.

propostas encontradas, implementando em um dos *kernel* do *Nas Parallel Benchmarnk*[1] (NPB) e executados em um ambiente virtualizado e controlado de cluster HPC. E então avaliadas métricas de overhead introduzido pela maleabilidade, impacto no tempo total de execução, esforço para adaptação da solução na aplicação escolhida e estabilidade do sistema após múltiplos ciclos de expansão e redução de recursos. Por fim, com os resultados obtidos, a melhor *runtime* encontrada para suporte à maleabilidade, será incorporada em uma aplicação que está no estado da arte. Espera-se também contribuir para o avanço da discussão sobre maleabilidade em MPI, fornecendo evidências experimentais que possam servir de base para adoção de soluções maleáveis em ambientes de produção HPC.

Referências

Feitelson, D. G. and Rudolph, L. (1996). Toward convergence in job schedulers for parallel supercomputers. In Feitelson, D. G. and Rudolph, L., editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–26, Berlin, Heidelberg. Springer Berlin Heidelberg.

MPI Forum (1997). Mpi-2: Extensions to the message-passing interface. Acesso em: 25 fev. 2026.

MPI Forum (2026). Mpi 4.0. Acesso em: 25 fev. 2026.

Streubel, M. (2020). Dynamic resource management using mpi sessions on p4est. Advisor: Martin Schreiber. Submission date: September 13, 2020.

[1]https://www.nas.nasa.gov/software/npb.html