Capítulo 1

Desenvolvimento de Aplicações Paralelas Adaptativas: Uma Visão de Duas Décadas (2001-2021)

Guilherme Galante
Universidade Estadual do Oeste do Paraná
Cascavel, Brasil

Rodrigo da Rosa Righi
Universidade do Vale do Rio dos Sinos
São Leopoldo, Brasil

Resumo

A evolução das arquiteturas paralelas aponta para ambientes dinâmicos nos quais o número de recursos ou configurações disponíveis pode variar durante a execução das aplicações. Isso pode ser facilmente observado em grades e nuvens, mas também pode ser explorado em clusters e arquiteturas de multiprocessadores. Ao longo de mais de duas décadas, várias iniciativas de pesquisa abordaram a exploração dessa característica por aplicações paralelas, permitindo o desenvolvimento de sistemas adaptativos que podem reconfigurar o número de processos/threads e seu mapeamento para processadores para lidar com cargas de trabalho variáveis e mudanças no disponibilidade de recursos no sistema. Nesse contexto, o objetivo deste documento é apresentar o estado da arte sobre adaptabilidade em pontos de vista de recursos e aplicações que vão desde arquiteturas de memória compartilhada, clusters e grades, até recursos baseados em névoa e nuvem. Uma análise abrangente das principais iniciativas de pesquisa em aplicações paralelas adaptativas pode fornecer aos leitores compreensão dos conceitos essenciais sobre a evolução da área. Além disso, esta revisão pode ajudar estudantes e pesquisadores a identificar as lacunas essenciais e oportunidades de tendências nesta área de investigação.

1.1. Introdução

A evolução das arquiteturas paralelas aponta para ambientes dinâmicos nos quais o número de recursos ou configurações disponíveis pode variar durante a execução das aplicações.

Essa variação de recursos pode ser observada em arquiteturas como grades computacionais e nuvens, assim como no nível de arquiteturas com múltiplos núcleos. Além disso, esses aspectos dinâmicos também podem aparecer no contexto das aplicações, nas quais processos (ou threads) obtêm cargas de trabalho desbalanceadas ou não uniformes ao longo do tempo. O mesmo pode acontecer com a comunicação de rede e acesso à memória, onde os padrões de uso podem mudar com o tempo, sendo muitas vezes dependente de arquivos de entrada ou entradas do usuário. Neste contexto, o desenvolvimento aplicações paralelas precisa acompanhar esta evolução e as novas tendências [Cera 2011].

[Feitelson and Rudolph 1996] classificam as aplicações paralelas de acordo com seu nível de adaptabilidade, categorizando-as em *rígidas, moldáveis, evolutivas e maleáveis*. Nas aplicações *Rígidas*, o número de recursos permanece constante ao longo da execução. Assim, eles devem ser executados usando nem mais nem menos recursos do que o especificado. As aplicações *moldáveis* têm alguma flexibilidade em termos dos recursos necessários. Os recursos são definidos no início da execução e a aplicação se adapta a esses recursos. Por sua vez, nas aplicações *evolutivas* e *maleáveis*, a quantidade de recursos pode variar durante as execuções. A diferença entre essas duas classes é que, no caso de aplicações *evolutivas*, a reconfiguração dos recursos começa a partir da própria aplicação. Em contraste, nas *maleáveis*, os recursos são controlados por um sistema subjacente (por exemplo, planejador, gerenciador externo ou tempo de execução). Considerando que a distinção entre aplicações evolutivas e maleáveis pode ser às vezes imprecisa, o termo *aplicações adaptativas* é comumente usado na literatura para se referir às duas classes acima mencionadas [Kalé et al. 2002].

Permitir que aplicações paralelas operem com alocação dinâmica de recursos pode apresentar vários benefícios. Dentre eles, destacam-se as melhorias no desempenho e na eficiência de aplicações [Galante and da Rosa Righi 2017]. Paralelamente a essas métricas, o custo e a tolerância a falhas também podem ser considerados na abordagem de aplicações de HPC dinâmicas. Por exemplo, um aplicação pode iniciar com uma quantidade padrão de recursos e, durante sua execução, novos recursos podem ser alocados a ela de acordo com suas demandas. Quando os recursos não são mais necessários, eles podem ser liberados, disponibilizando-os para outras tarefas na fila (em um cluster ou nuvem). Esses mecanismos de alocação flexível também podem melhorar o rendimento do sistema, o tempo de espera e o tempo de resposta de outras tarefas [Sudarsan and Ribbens 2007]. A adaptabilidade também é adequada para aplicações cujos requisitos de recursos não podem ser determinados com precisão com antecedência e no balanceamento de carga [Prabhakaran et al. 2014]. Outro exemplo pode ser apresentado no contexto da computação em nuvem e névoa, em que a alocação dinâmica de recursos é chamada de *elasticidade* [Galante and Bona 2012, Herbst et al. 2013]. As aplicações podem usar várias máquinas virtuais (VMs) ou contêineres que são instanciados ou redimensionados dinamicamente para moldar-se aos requisitos da aplicação por meio do modelo de pagamento pelo uso. Observando esses cenários, os benefícios de adaptabilidade para a área de computação paralela são claros. De acordo com *The International Exascale Software Project Roadmap* [Dongarra et al. 2011], o provisionamento dinâmico de recursos é considerado um aspecto essencial para a exploração da computação em exaescala.

Neste contexto, propõem-se uma revisão do estado da arte no âmbito das aplicações paralelas adaptativas. Mais especificamente, apresenta-se como a adaptabilidade pode ser explorada em aplicações e arquiteturas com vários núcleos de processamento, clusters e grades, incluindo também computação na névoa e em nuvem. Essa análise abrangente das principais iniciativas de pesquisa em aplicações paralelas adaptativas pode fornecer aos leitores uma compreensão dos conceitos essenciais da evolução da área. Além disso, essa pesquisa ajudará indivíduos e pesquisadores (experientes ou não) a identificar as questões cruciais e as lacunas para uma pesquisa mais aprofundada.

1.2. Aplicações Adaptativas: Estado da Arte

Esta seção tem o objetivo de apresentar e discutir as soluções propostas no estado da arte para o desenvolvimento de aplicações adaptativas em diferentes arquiteturas paralelas.

1.2.1. Arquiteturas de Memória Compartilhada

Em um sistema multiprocessador de memória compartilhada, vários processadores (ou núcleos) acessam um único espaço de endereçamento de memória. Esta é uma configuração muito conveniente do ponto de vista da programação, já que os dados gerados por um processador e armazenados na memória principal são imediatamente acessíveis pelos outros processadores/núcleos [Kale 2020]. Uma aplicação paralela executada em uma arquitetura de memória compartilhada geralmente usa várias threads. Um programa *multithread* pode ser escrito em diferentes linguagens de programação usando muitas bibliotecas e frameworks diferentes, que devem fornecer instruções para iniciar threads, atribuir trabalho a elas e coordenar seus acessos aos dados. Exemplos de bibliotecas e frameworks populares incluem Pthreads, OpenMP, CILK e Threading Building Blocks.

Normalmente, o número de threads usado é fornecido na execução da aplicação por meio de um parâmetro de linha de comando ou uma variável de ambiente, e permanece durante seu tempo de vida. A adaptabilidade geralmente não é explorada [Grelck 2015]. No entanto, as aplicações multithread podem ser convertidas ou implementadas como aplicações adaptativas para obter algum benefício. Nesse sentido, alguns trabalhos abordam essa questão na literatura.

[Hungershöfer and Wierum 2002] apresentam um gerenciador de paralelismo da aplicação (*application parallelism manager* - APM) para determinar a atribuição dinâmica de processadores a tarefas com base no desempenho estimado. As arquiteturas alvo são SMPs multiusuário executando várias aplicações simultaneamente que podem enfrentar conflitos de recursos entre os usuários. O objetivo geral é garantir que o sistema não fique ocioso nem sobrecarregado. O APM é implementado como um único servidor que coleta informações de aplicações, calcula uma distribuição de recursos adequada e envia recursos atribuídos a cada aplicação conectada. Uma biblioteca com métodos de comunicação deve ser vinculada à aplicação para permitir a comunicação com o servidor via sockets TCP. Presume-se que os aplicativos são maleáveis.

O trabalho de [Utrera et al. 2004] objetiva fornecer maleabilidade a programas MPI em multiprocessadores. Os autores apresentam uma técnica chamada *Folding by JobType* (FJT), na qual os jobs paralelos são executados com uma pequena quantidade de recursos (*folded* - dobrados) e são expandidos quando os recursos ficam disponíveis.

As operações de dobramento são gerenciadas pelo runtime do sistema, sendo transparente para as aplicações. [Suleman et al. 2008] propõem o Feedback-Driven Threading (FDT), um framework para controlar o número de threads usando informações coletadas em tempo de execução. O FDT amostra uma pequena fração dos kernels paralelizados para estimar o comportamento da aplicação (sincronização de dados e saturamento do barramento off-chip). Com base nessas informações, ele estima o número de threads nos quais o desempenho do kernel satura o barramento, e então, ajusta o número de threads da aplicação.

Em sua tese, [McFarland 2011] descreve o sistema RSM, que suporta aplicações OpenMP em plataformas de memória compartilhada. O RSM inclui um runtime que coleta informações de desempenho e toma decisões de redimensionamento, e uma biblioteca de comunicação para permitir que as aplicações se comuniquem com o daemon RSM para realizar o redimensionamento de recursos. Além desse trabalho, [Gordon and Lu 2011] apresentam o Elastic Phoenix, uma implementação MapReduce para sistemas de memória compartilhada com suporte a trabalhos maleáveis. A estrutura permite que threads sejam adicionados ou removidos durante a execução de uma tarefa. Com o Elastic Phoenix, se mais recursos estiverem disponíveis (como em um sistema de computação em nuvem elástica), eles poderão ser adicionados dinamicamente a uma tarefa existente. Se os recursos forem necessários para uma outra aplicação, eles também poderão ser removidos de uma tarefa existente.

[Galante and Bona 2014] propõem o Elastic OpenMP, um mecanismo para fornecer alocação dinâmica de processadores virtuais (VCPUs) para aplicações OpenMP executadas em máquinas virtuais com múltiplos núcleos. A API OpenMP original foi estendida recebendo modificações para permitir o provisionamento dinâmico de recursos. As extensões de alocação dinâmica de recursos são baseadas em adaptações de diretivas OpenMP e na adição de um conjunto de rotinas de nível de usuário à API OpenMP. A Figura 1.1 ilustra a interação API-Runtime-Cloud. Pode-se observar no código fonte um diretiva *parallel* que cria as threads e solicita a alocação das VCPUs, enviando uma mensagem para o runtime. O runtime trata a mensagem e interage com a plataforma de nuvem solicitando os recursos por meio de uma interface fornecida pelo provedor de nuvem.

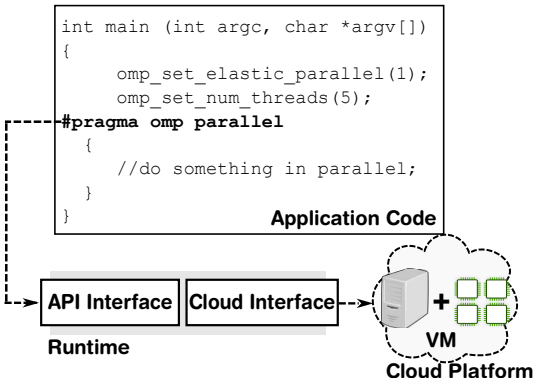


Figura 1.1. Elastic OpenMP: Interações entre a API, Runtime e Plataforma da Nuvem

[Grelck 2015] descreve um sistema de tempo de execução orientado a filas, no qual tarefas S-NET refinadas são mapeadas automaticamente para o número dinamicamente variável de threads de trabalho (ou núcleos) efetivamente disponíveis. O runtime aloca dinamicamente os recursos de execução para uma aplicação em execução e monitora continuamente a adequação dos recursos considerando as demandas do aplicativo e a carga do sistema.

[Creech 2015] apresenta o SCAF, um runtime para suportar a alocação dinâmica de recursos para aplicações maleáveis OpenMP com base na eficiência. O SCAF calcula no início de uma seção paralela a eficiência de cada aplicação para estimar a alocação de recursos. O objetivo geral é melhorar a soma de speedups das aplicações que compartilham recursos de máquina. [Georgakoudis et al. 2017] apresentam uma solução semelhante, SCALO, cuja intenção é otimizar o *throughtput* das aplicações paralelas que compartilham os recursos de um nó. O SCALO monitora as aplicações em tempo de execução para avaliar sua escalabilidade e adaptar o paralelismo de cada programa (número de threads). [Cho et al. 2018] desenvolveram o NuPoCo, um framework para gerenciar automaticamente o paralelismo de aplicações paralelos OpenMP co-alocados em sistemas NUMA. O modelo de desempenho da estrutura prevê a utilização de núcleos de CPU e memória para aplicações paralelas co-alocados; em seguida, essas informações são usadas para determinar o grau de paralelismo para todas as cargas de trabalho em execução com o objetivo de maximizar a utilização dos recursos do sistema.

Além dos esforços para fornecer runtimes e frameworks, alguns trabalhos abordam a alocação dinâmica de recursos para aplicações específicas em arquiteturas de memória compartilhada. [Pagani et al. 2016] e [Dominico et al. 2018] focam em mecanismos elásticos de alocação de núcleos de processamentos e computação para bancos de dados. [Catalán et al. 2019] introduz uma implementação maleável em nível de thread para uma biblioteca de álgebra linear (BLAS). [Libutti et al. 2020] fornecem algumas modificações no TensorFlow para introduzir maleabilidade.

1.2.2. Clusters

[Stallings 2017] define um cluster como um conjunto de computadores interconectados trabalhando juntos como um recurso de computação unificado que pode criar a ilusão de ser uma entidade única. Em um cluster, cada computador é chamado de nó. Para construir um cluster de computador, os nós individuais devem ser conectados a uma rede para permitir a comunicação entre entre esses nós. Uma camada de software pode então ser usado para unir os nós e formar um cluster.

O modelo de programação para esse sistema de memória distribuída geralmente é um modelo de troca de mensagens no qual as mensagens são usadas para a troca de informações entre os computadores. Message Passing Interface (MPI) é o método *de facto* para desenvolver aplicações em clusters computacionais. O padrão MPI original (versão 1) exigia que o número de processos fosse fixado na inicialização do programa. O suporte a processos dinâmicos foi incluído na versão 2.0 do padrão. Devido aos custos de desempenho e limitações dos recursos, o suporte a processos dinâmicos do MPI não tem sido amplamente utilizado pelos desenvolvedores de aplicações [Comprés et al. 2016].

Nesse contexto, vários esforços de pesquisa tem como foco a adaptabilidade de programas MPI.

[Huang et al. 2004] definem o Adaptive MPI (AMPI), uma implementação do MPI sobre o runtime do Charm++. AMPI implementa processos MPI como threads de nível de usuário (objetos Charm++). A maleabilidade é alcançada iniciando as aplicações com uma granularidade de processo muito fina e contando com um balanceador de carga para atribuir os objetos aos recursos físicos. Os programas AMPI recebem informações sobre a disponibilidade de processadores de um agendador de tarefas adaptável; em seguida, o runtime usa a migração de objetos para adaptar a aplicação a novos recursos.

ReSHAPE [Sudarsan and Ribbens 2007] é um framework para o desenvolvimento de aplicações MPI iterativas e maleáveis que usa dados de desempenho coletados em tempo de execução para dar suporte a ações de reconfiguração. As aplicações executadas usando o ReSHAPE podem ser expandidos para aproveitar os processadores livres ou reduzidos para acomodar uma aplicação de alta prioridade sem serem suspensos. A reconfiguração dinâmica é fornecida por um módulo de remapeamento que usa a interface de gerenciamento de processos dinâmicos do MPI-2 para gerar novos processos. O ReSHAPE é mais adequado para aplicações em que dados e processamento são distribuídos de maneira relativamente uniforme entre os processadores.

A PCM (Process Checkpointing and Migration) [El Maghraoui et al. 2009] é uma biblioteca/runtime que permite que programas SPMD MPI se reconfigurem para se adaptarem aos processadores disponíveis. A estratégia empregada adota procedimentos para dividir/mesclar processos para diminuir/aumentar a granularidade e migrar esses processos de acordo com a disponibilidade de recursos. A disponibilidade do processador é gerenciada por um agente que monitora o hardware. As ações adaptativas são realizadas sem intervenção do usuário, mas é necessário que o programador instrumente o código-fonte com primitivas PCM. [Kim et al. 2011] exploram a maleabilidade em modelos acoplados paralelos. Modelos acoplados paralelos evoluem seu estado resolvendo cada módulo de aplicação em seu respectivo conjunto de processadores (cohorts). Nesse contexto, os autores apresentam o Maleable Model Coupling Toolkit (MMCT) para construir modelos acoplados paralelos cuja configuração dos cohorts dos módulos é ajustada automaticamente em tempo de execução por meio de decisões tomadas por um gerenciador de balanceamento de carga. Recursos do MPI-2 são usados para gerenciar a criação de processos dinâmicos.

[Cera 2011] apresenta uma abordagem para fornecer maleabilidade em aplicações paralelas usando a interface de gerenciamento de processo dinâmico do MPI-2. O trabalho se concentra na criação de processos dinâmicos para fornecer adaptabilidade a processadores voláteis (aplicativos maleáveis), bem como necessidades imprevisíveis (aplicativos de tarefas explícitas). Além disso, foram descritas as interações necessárias entre as aplicações MPI e o Sistema de gerenciamento de recursos (OAR) para troca de informações sobre a disponibilidade dos processadores. Ainda na ideia de estender o modelo MPI, cita-se o Flex-MPI [Martín et al. 2015], uma extensão MPI para permitir a maleabilidade e a reconfiguração dinâmica de aplicações iterativas SPMD. O Flex-MPI reconfigura automaticamente a aplicação para ser executada com o número de processos necessários para aumentar o desempenho de modo que a aplicação seja concluída dentro

de um intervalo de tempo especificado. Um modelo de previsão computacional é usado para avaliar múltiplos cenários de reconfigurações potenciais e escolher aquele que é previsto para satisfazer melhor o objetivo de desempenho.

[Comprés et al. 2016] apresentam o framework de maleabilidade Elastic MPI. Neste trabalho, o gerenciador de recursos Slurm e a biblioteca MPICH são estendidos com novas funcionalidades para lidar com a reconfiguração de aplicações paralelas. O Slurm foi estendido para gerenciar a criação e exclusão de processos MPI enquanto lida com a alocação de recursos. Novas funções também foram adicionadas à biblioteca MPI. Quando essas funções são utilizadas, a aplicação é inicialmente definida como maleável, e seus processos verificarão periodicamente se o Slurm iniciou uma reconfiguração. [Lemarinier et al. 2016] abordam a maleabilidade usando duas abordagens diferentes: na primeira, os autores usam um mecanismo tradicional de checkpoint/restart, aproveitando a biblioteca Scalable Checkpoint/Restart for MPI (SCR), para reexecutar um trabalho com um novo número de processos após salvar seu estado. A segunda abordagem é baseada no User Level Failure Migration (ULFM), um conjunto de funções e estruturas de dados para adicionar recursos de tolerância a falhas a programas MPI existentes. As aplicações são interrompidas, o checkpoint é realizado e, em seguida, reinicia-se as aplicações com uma nova quantidade de recursos.

[Iserte et al. 2018] propõem um framework, denominado Dynamic Management of Resources (DMR), para fornecer maleabilidade ao MPI, baseando-se no MPI, OmpSs e Slurm. O DMR é implementado aproveitando o modelo de programação OmpSs, que foi estendido para lidar com reconfiguração dinâmica. Ele interage com o Slurm por meio de chamadas de API. Em seguida, o gerenciador de recursos inspeciona o status global do sistema para decidir se deve iniciar a ação de redimensionamento. O DMR expande e reduz os trabalhos rapidamente, reatribuindo os recursos subjacentes, gerando novos processos MPI, redistribuindo os dados entre os processos e retomando a execução. Posteriormente, os autores apresentam um estudo sobre o projeto, implantação e avaliação de aplicações maleáveis em GPU usando o DMR [Iserte and Rojek 2020]. O Dynamic Resource Ownership Management (DROM) [D'Amico et al. 2018] também é baseado na integração da API e Slurm. Ele fornece um canal de comunicação para permitir que o escalonador se comunique com as aplicações para adaptar os recursos de computação. Além de suportar MPI para aplicações em memória distribuída, DROM permite maleabilidade no nível do nó computacional, alterando também o número de threads que as aplicações OpenMP ou OmpSs estão usando.

Além dos esforços para desenvolver frameworks e arquiteturas com foco em MPI, alguns trabalhos abordam ainda a adaptabilidade em outros modelos e níveis de aplicação. [Batheja and Parashar 2003] desenvolveram um framework para computação em cluster adaptável e oportunista. O framework visa aplicações master-slave e é construído nas tecnologias Java e JavaSpaces. [Gupta et al. 2014] apresentam um mecanismo implementado sobre CHARM++ para permitir a maleabilidade em um runtime paralelo usando migração de tarefas, balanceamento de carga, checkpoint/restart e memória compartilhada Linux. O trabalho de [Fox et al. 2017] apresenta o Elastic-HPC (E-HPC), um framework elástico para gerenciamento de recursos para workflows científicos em sistemas HPC. A estrutura usa a checkpoint/restart como o mecanismo subjacente para migrar a execução do workflow no conjunto dinâmico de recursos.

Para tirar proveito do uso de tarefas maleáveis em ambientes de cluster compartilhado, os escalonadores e os gerenciadores de recursos também precisam ser adaptativos. Nesse contexto, alguns trabalhos abordam esse assunto. [Klein and Perez 2011] propõem o CooRMv2, um gerenciador de recursos que fornece escalonamento eficiente de aplicações maleáveis e evolutivas. [Prabhakaran et al. 2014] apresentam uma extensão do sistema Torque/Maui que permite alocações dinâmicas. O Elastic Job Bundling (EJB) é uma camada de software que opera entre usuários finais de aplicações paralelas e sistemas de batch HPC [Liu and Weissman 2015]. O objetivo do EJB é reduzir o tempo de resposta de aplicações paralelas. O EJB decompõe dinamicamente uma tarefa grande em tarefas menores para reduzir o tempo de espera e permite que a aplicação se expanda em várias subtarefas enquanto obtém progresso contínuo.

Também é possível encontrar na literatura o projeto e desenvolvimento de aplicações adaptativas em clusters. [Leopold et al. 2006] descrevem uma versão maleável do WaterGAP, uma aplicação científica que simula a disponibilidade global de água usando recursos do MPI-2. [Sudarsan et al. 2009] investigam o uso do ReSHAPE para redimensionar códigos de computação científica. Como caso de teste, o código de simulação de dinâmica molecular (MD) LAMMPS é usado. [Mo-Hellenbrand et al. 2017] apresentam um simulador de tsunami maleável usando o Elastic MPI. [Mascagni et al. 2019] descrevem o caso particular de usar a API DMR para gerar uma versão maleável do HPG, um sequenciador genômico não iterativo de memória distribuída. O DMR também é usado para incluir maleabilidade no Algoritmo de Transporte de Advecção Definida Positiva Multidimensional (MPDATA) [Iserte and Rojek 2020]. [Spenke et al. 2019] mostram uma implementação adaptativa do OGOLEM, um framework para otimização global de problemas químicos, baseado em Java-RMI.

1.2.3. Grids

Um grid, ou grade computacional, pode ser definido como um sistema unificado de memória distribuída composto por uma coleção de muitos nós/processadores distribuídos em uma ampla área geográfica com o objetivo de obter computação de alto desempenho e compartilhamento de recursos [Wilkinson 2009]. Considerando que a Internet ou redes de baixa velocidade acoplam essas infraestruturas distribuídas, as grades são mais adequadas para problemas paralelos de granularidade grossa, compostos principalmente por tarefas independentes, que são gerenciadas e executadas por meio de sistemas de workflow ou ambientes de programação paralela [Foster et al. 2008]. Grids são sistemas dinâmicos que podem acomodar nós que entram e saem ao longo do tempo. Assim, é altamente desejável reconfigurar as aplicações em execução em resposta às mudanças no ambiente (deliberadas ou causadas por falhas).

No framework GrADS [Kennedy et al. 2002], cada aplicação tem um gerenciador que monitora o desempenho para alcançar a QoS desejada. O componente Monitor de Contrato identifica violações e inicia o reescalonamento da aplicação (checkpoint/restart). As ações de reecalonamento incluem substituir recursos específicos, redistribuir as tarefas da aplicação nos recursos atuais e adicionar ou remover recursos. Além do referido trabalho, [Vadhiyar and Dongarra 2003] apresentam uma biblioteca de checkpoint chamada SRS (Stop Restart Software) e um Runtime Support System (RSS), visando aplicações MPI iterativas. A maleabilidade é implementada por um mecanismo de checkpoint/restart

que permite que uma aplicação seja verificada e, em seguida, reiniciada com um número diferente de processadores. Nesse trabalho, o objetivo da maleabilidade é garantir tolerância a falhas para aplicações executando em uma grade e ainda permitir o uso de recursos livres adicionais.

Outra solução baseada em checkpoint/restart é o Performance Control System (PerCo) [Mayes et al. 2005]. O sistema monitora o progresso das aplicações e as realoca para otimizar o desempenho. O alvo são aplicações fracamente acopladas que podem ser realocadas e reiniciadas com um número diferente de processos. Ainda no cenário de tolerância a falhas, [Wrzesinska et al. 2005] apresentam um sistema que suporta tolerância a falhas, maleabilidade e migração para aplicações de divisão e conquista. Os mecanismos são implementados usando Satin, um sistema de divisão e conquista baseado em Java projetado para ambientes de grade [Van Nieuwpoort et al. 2010]. No Satin, o trabalho é distribuído entre os processadores por meio de roubo de trabalho (*workstealing*). Adicionar uma nova máquina implica roubar trabalhos de outras máquinas. Quando uma máquina é removida ou para, os trabalhos são colocados de volta nas filas de trabalho para que eventualmente sejam recomputados. O Satin também pode adaptar aplicativos automaticamente para reagir a mudanças nas condições de recursos, como CPUs sobrecarregadas ou links de comunicação lentos.

No ambiente ASSIST [Aldinucci et al. 2006] aplicações paralelas são implementadas usando parmods (módulos paralelos). Quando um parmod é executado, um processo de monitoramento é executado para coletar informações de desempenho. Se um desempenho insatisfatório for detectado, um processo gerenciador é informado e uma ação corretiva é planejada. Neste caso, a política que pode ser adotada requer o incremento do número de recursos empregados na execução do parmod. Assim, o parmod é parado assim que atinge um ponto de sincronização e é reiniciado usando a adição de recursos. Além da iniciativa mencionada, o Dynaco (Dynamic Adaptation for Components) é um framework que ajuda no desenvolvimento e implementação de componentes dinamicamente adaptáveis [Buisson et al. 2007]. Os aplicativos Dynaco realizam adaptações baseadas em eventos, de acordo com a dinamicidade dos processadores. Quando uma aplicação observa que novos processadores estão disponíveis, ela pode aumentar o grau de paralelismo gerando e inicializando novos processos e redistribuindo dados.

[Klemm et al. 2009] apresentam uma abordagem para paralelizar e migrar aplicações OpenMP (mais precisamente, OpenMP para Java) entre clusters de uma grade. Um usuário inicia uma aplicação em um cluster arbitrário na grade. Um ponto de verificação é criado quando a fatia de tempo (alocada nesse cluster) está prestes a ser excedida. A aplicação pode migrar para outro cluster ou reiniciar o sistema atual com uma nova reserva. A paralelização permite que a próxima reserva de recurso local solicite menos ou mais CPUs, dependendo da carga geral do sistema. Assim, a aplicação deve explorar totalmente os recursos livres e adaptar as threads OpenMP aos elementos de processamento disponíveis.

[Ribeiro et al. 2013] estendem o EasyGrid de modo de transformar aplicativos MPI moldáveis tradicionais em versões autonômicas maleáveis. A aplicação maleável EasyGrid é capaz de, em alguns pontos de reconfiguração, ajustar automaticamente sua granularidade e alocação de processo de acordo com o poder computacional disponível.

1.2.4. Computação em Nuvem

No âmbito da nuvem, a elasticidade aparece como principal característica para auxiliar no desempenho de aplicações HPC. A Figura 1.2 ilustra os dois mecanismos de funcionamento: (a) é mais simples e usado principalmente em provedores de nuvem pública como Amazon e Google; (b) precisa de modelos de previsão para entender antecipadamente quando os limites podem ser ultrapassados. Além disso, em (b), a temática de reconhecimento de padrões pode ser usada para detectar o comportamento da aplicação HPC, iniciando novos contêineres ou máquinas virtuais de forma que sejam totalmente entregues e integrados antes de haver qualquer problema de carregamento da aplicação.

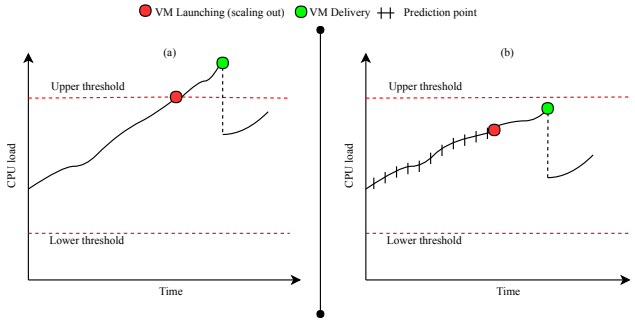


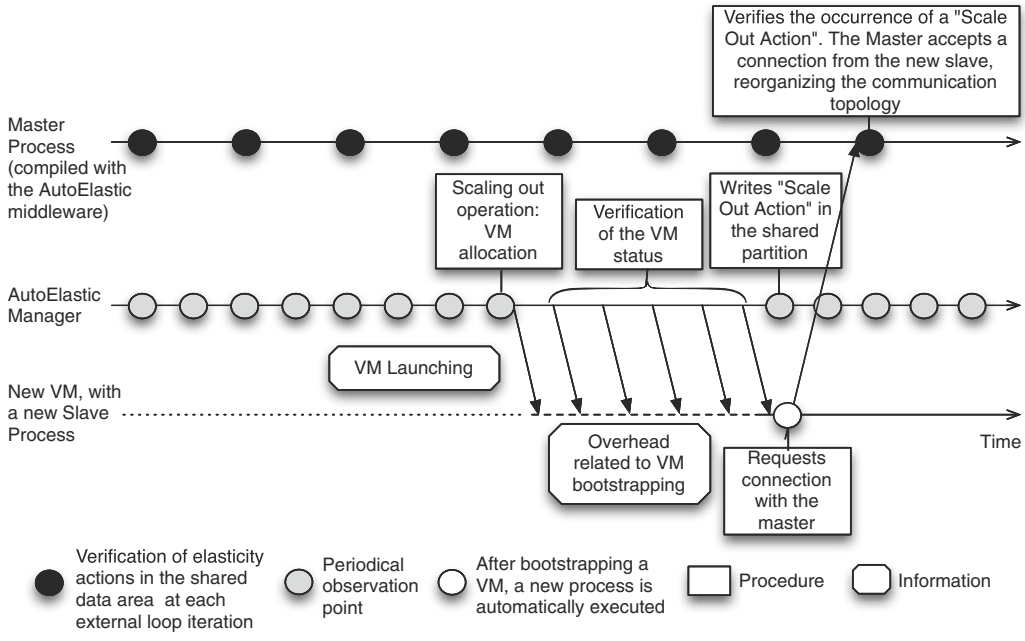
Figura 1.2. Tratamento da elasticidade em duas possibilidades: (a) reativa, uma vez que uma ação é tomada quando um threshold predefinido é atingido; (b) proativa, usando dados de predição para antecipar eventuais problemas ou irregularidades quando executando aplicações HPC.

Em [Kehrer and Blochinger 2020], os autores abordam os desafios de elasticidade para aplicações de pesquisa em árvore paralela, fornecendo um modelo denominado Equilibrium. Os autores mostram que aplicações como branch-and-bound e backtracking search não são executados de forma otimizada com recursos rígidos, de modo que a elasticidade pode fornecer desempenho e eficiência. Eles discutem como construir um controlador de elasticidade que raciocina o comportamento de dimensionamento de um sistema paralelo em tempo de execução e adapta dinamicamente o número de unidades de processamento de acordo com o custo definido pelo usuário e os limites de eficiência. Os autores reconhecem que uma compreensão detalhada do comportamento de dimensionamento de uma aplicação é uma questão fundamental sobre o qual os mecanismos de controle de elasticidade devem ser construídos. Eles se concentraram no modelo de pool de tarefas distribuídas, onde a adaptabilidade da aplicação é direta, ou seja, eles não precisam ser reestruturados para se beneficiar com a elasticidade dos recursos da nuvem.

[Rodrigues et al. 2018, Rodrigues et al. 2017] descrevem o SelfElastic, uma solução que combina abordagens de elasticidade pró-ativa e reativa para executar aplicações HPC. Eles argumentam que temos pelo menos um problema ao usar apenas uma abordagem ativa ou proativa: a necessidade de uma experiência anterior do usuário, falta de manipulação de picos de carga, conclusão de parâmetros ou design para infraestrutura específica e conjunto de carga de trabalho. O SelfElastic apresenta uma arquitetura de elasticidade de loop de controle fechado que adapta os valores dos limites inferior e superior em tempo de execução como um modelo sem parâmetros. O SelfElastic oferece

elasticidade híbrida por meio da técnica Live Thresholding, portanto, valores de limite de auto-organização e alocação de recursos para oferecer uma solução competitiva em níveis de desempenho e custo.

[da Rosa Righi et al. 2016] apresentam o AutoElastic, um modelo de elasticidade de nível de PaaS para HPC na nuvem. Sua abordagem diferencial consiste em fornecer elasticidade para aplicações de alto desempenho sem intervenção do usuário ou modificação do código-fonte. As contribuições científicas do AutoElastic são: (i) uma abordagem baseada no envelhecimento para alocação de recursos e ações de desalocação para evitar reconfigurações desnecessárias de VMs (*thrashing*) e (ii) assincronismo na criação e encerramento de VMs de forma que a aplicação não precisa aguardar a conclusão desses procedimentos. Em (i), os autores usaram a técnica de média móvel exponencial para suavizar os dados de carga da CPU rada, desencadeando ações de reorganização de recursos mais decisivas. Em (ii), eles fornecem um protocolo de interação entre o processo mestre e o gerenciador de elasticidade (ver Figura 1.3). O gerenciador informa o processo mestre quando novos recursos estão disponíveis. Além disso, ao detectar uma situação de subcarga, o gerenciador informa o IP de uma máquina virtual ao processo mestre, que desconecta os escravos dessa VM da aplicação. Depois disso, o gerenciador está livre para consolidar a VM de destino, matando todos os escravos que executam nela.



/2/2

Figura 1.3. Elasticidade assíncrona através da orquestração de aplicação (com o processo mestre) e o gerente de elasticidade. Ações de elasticidade podem ser realizadas ser que seja necessária a parada da aplicação até a entrega de novos recursos.

Historicamente, os projetistas de aplicações ajustam aplicações simultâneas para hardware e plataformas específicos. No entanto, essas abordagens não são viáveis em plataformas de nuvem, pois as aplicações podem ser implantadas em várias plataformas e os ambientes operacionais podem variar em cada implantação. Nesse contexto,

[Rajan and Thain 2017] argumentam e demonstram que aplicações concorrentes em plataformas de nuvem devem ser autoajustáveis. Primeiro, eles mostram que os aplicações devem incorporar um modelo de overheads de operação. Em segundo lugar, eles mostram que as aplicações devem determinar seus requisitos de recursos e ajustar sua operação às condições operacionais usando estimativas do modelo. Para testar as ideias mencionadas, os autores construíram duas aplicações de autoajuste, E-Sort e E-MAKER. Eles demonstram sua capacidade de obter alta eficiência de custo determinando a escala correta de partições e recursos a serem usados para operação e adaptando seu comportamento de acordo com as características do ambiente implantado.

[Galante and Erpen De Bona 2015] argumentam que a maioria das soluções são ineficientes em fornecer elasticidade para aplicações científicas, uma vez que não podem considerar a estrutura interna e o comportamento das aplicações. Eles apresentam uma abordagem para explorar a elasticidade em aplicações científicas, em que o controle de elasticidade é embutido no código-fonte da aplicação e construído usando primitivas de elasticidade. Essa abordagem permite que a aplicação solicite ou libere seus recursos, levando em consideração o fluxo de execução e os requisitos de tempo de execução. Para apoiar a construção de aplicações elásticas usando a abordagem apresentada, eles desenvolveram o framework Cloudine. O Cloudine fornece todos os componentes necessários para construir e executar aplicações científicas elásticas. A eficácia do framework é demonstrada nos experimentos em que a plataforma é usada com sucesso para incluir novos recursos para aplicações existentes, estender as funcionalidades de outras estruturas de elasticidade e adicionar suporte de elasticidade a bibliotecas de programação paralela.

[Raveendran et al. 2011] argumentaram em 2011 que havia uma tendência clara de usar recursos de nuvem na comunidade científica ou de HPC, com uma atração crítica da nuvem sendo a elasticidade que ela oferece. Segundo eles, na execução de aplicações HPC em um ambiente de nuvem, será desejável explorar a elasticidade dos ambientes de nuvem e aumentar ou diminuir o número de instâncias que uma aplicação é executada durante a execução da aplicação para atender às restrições de tempo e/ou custo. Em [Raveendran et al. 2011], descreve-se um trabalho inicial com o objetivo de tornar os aplicações MPI existentes elásticos para uma estrutura de nuvem. Considerando as limitações das implementações MPI atualmente disponíveis, eles suportam a adaptação encerrando uma execução e reiniciando um novo programa em várias instâncias. Os componentes de seu sistema idealizado incluem uma camada de decisão que considera as restrições de tempo e custo, uma estrutura para modificar programas MPI e suporte de tempo de execução baseado em nuvem que pode permitir a redistribuição de dados salvos e apoiar a alocação automatizada de recursos e reinicialização de aplicações em um número diferente de nós.

1.2.5. Computação em Névoa

Observa-se que a computação em névoa é amplamente explorada no escopo IoT (Internet of Things), onde o desempenho é visto na maioria das vezes como redução da latência da rede. Os serviços essenciais podem ser colocados na névoa para fornecer uma resposta pontual aos aplicações do usuário final, em vez de acessar a nuvem para computar insights de IA. Desta forma, [Yin et al. 2018] propuseram o uso de um avaliador de solicitação como o primeiro módulo nos nós de Fog Computing, permitindo avaliar a complexidade

de uma tarefa com base em seu tempo de computação calculado para decidir onde tal tarefa devem ser endereçados para atender aos seus requisitos. O avaliador da solicitação decide se a tarefa não pode ser endereçada localmente no dispositivo de borda. Caso contrário, eles são mapeados para a névoa ou nuvem. O escalonador de tarefas avalia a demanda de recursos de uma tarefa e o período de névoa é o limite de recursos calculado para determinar a alocação. Seu estudo se concentrou na alocação de recursos adequada para aderência de QoS e comunicação reduzida para servidores remotos, minimizando os riscos de congestionamento de rede. [Naha et al. 2020] usou uma abordagem de classificação de recursos - com base no processamento disponível, latência e largura de banda - para alocação dinâmica de recursos em arquiteturas Cloud-Fog como um meio de obter tempos de resposta ideais. O método de provisionamento de recursos incluiu a transmissão de dados como uma métrica para a decisão de provisionamento e uma abordagem hierárquica para minimizar a latência, os tempos de resposta e o congestionamento da rede priorizando os membros mais baixos da hierarquia sempre que possível.

[Chen et al. 2017], por outro lado, propôs uma arquitetura de quatro camadas para lidar com aplicações HPC baseados em IoT - consistindo em IoT, Middleware, Fog e Cloud - onde a maioria dos serviços são fornecidos pela nuvem inicialmente, e o mesmo é alocado para a névoa sob demanda. O middleware é responsável por receber tarefas de dispositivos IoT e realizar a classificação de trabalhos e agendamento de recursos em seu trabalho. A classificação do trabalho é baseada na privacidade de dados e nos requisitos de QoS. Tarefas sujeitas à privacidade de dados são atribuídas diretamente a um nó Fog local para garantir a segurança. Em contraste, tarefas não sensíveis à privacidade têm seus requisitos de QoS avaliados para determinar quais nós do sistema podem atender a QoS. Com base nos resultados da classificação, o middleware escalona tarefas para a Fog ou Cloud com base em uma avaliação de custo operacional realizada pelo agendador de recursos usando custos predefinidos. [Small et al. 2017] apresenta uma solução de middleware para orquestração baseada em microsserviços de aplicações em infraestruturas de IoT multicamadas como um meio de permitir a comunicação reduzida com servidores remotos. Sua solução orquestra a implantação de serviços nas camadas Cloud, Fog ou Mist com base nos requisitos dos serviços e nos recursos disponíveis nas camadas. A camada de nuvem usa VMs hospedadas em OpenStack, tendo a capacidade de inicializar VMs antecipadamente, reduzindo o tempo de implantação. Ao mesmo tempo, os serviços na névoa são fornecidos por contêineres instanciados sob demanda.

Muitos autores propõem diferentes tipos de configurações de hardware para permitir diferentes casos de uso do Fog até hoje. [He et al. 2018] descreve um modelo de arquitetura Fog multicamadas para permitir que tarefas analíticas complexas ocorram no nível Fog e reduzir a carga de solicitações direcionadas aos servidores Cloud para cenários de analítica de dados. O modelo multicamadas proposto é composto por duas camadas de nós Fog, A-Fog (ad-hoc) e D-Fog (dedicado), que operam em tarefas analíticas com base em sua complexidade [He et al. 2018]. O A-Fog era composto por dispositivos de baixo poder de computação, enquanto o D-Fog era formado por um cluster de servidores, permitindo a execução de análises de dados complexas sem um servidor em nuvem. A alocação de recursos é baseada no custo da utilidade como uma medida de decisão, e não na aderência ao QoS, como em alguns estudos anteriores.

Em vez de focar na alocação de recursos em Fog ou Cloud, Alsaffar propôs uma estratégia de colaboração entre Fog Computing e Cloud Computing para permitir a execução de tarefas mais complexas [Choi et al. 2016]. Em seu trabalho, um broker de Fog avalia as solicitações recebidas. Se seu nó não pude processá-lo dentro de seus SLAs, ele contata um servidor monitor de serviço localizado na nuvem, que verifica a disponibilidade da VM em todos os ambientes Cloud e Fog. Com base na disponibilidade de informações recuperadas do servidor do monitor de serviço, o intermediário Fog divide o trabalho em vários blocos para distribuição nas VMs disponíveis. Após a conclusão do processamento, cada VM remota retorna os dados para agregação ao intermediário Fog inicial.

[Al-khafajiy et al. 2019] propõem um sistema de colaboração Fog-2-Fog para permitir uma melhor utilização de recursos e distribuição de carga entre os nós Fog. O estudo projetou algoritmos para decisões de realocação de recursos usando múltiplas métricas de atraso, como atraso de serviço, atraso de propagação e atraso computacional [Al-khafajiy et al. 2019]. A decisão de quando descarregar o trabalho de um nó Fog para outro foi baseada em duas condições: 1. Avaliação se um ou mais serviços na fila perderiam seu prazo; 2. Comparação da taxa de chegada de serviço com a saída de serviço no nó. Nguyen et al. [Nguyen et al. 2020] aponta que meramente monitorar as métricas do servidor, como carga de CPU e consumo de memória, não é uma solução apropriada para lidar com a heterogeneidade de aplicações compreendidos pela IoT. Nessa nota, Nguyen propôs ElasticFog, uma estrutura construída em cima do Kubernetes para alocação dinâmica de recursos de aplicações baseados em contêiner em Fog Computing. ElasticFog monitora o tráfego de rede em cada local do nó Fog e usa isso como uma regra de afinidade no Kubernetes para melhorar as decisões de alocação de recursos. A avaliação da solução mostrou melhorias significativas na taxa de transferência e latência em favor do ElasticFog em comparação com o mecanismo de escalonamento padrão do Kubernetes. A Figura 1.4 ilustra essa arquitetura, onde pode-se ampliar o número de recursos de névoa de acordo com a demanda de IoT.

1.2.6. Discussão

Nesta seção, discute-se o estado da arte apresentado, abordando as principais características das soluções propostas para cada arquitetura. Inicialmente, discute-se aqui as soluções de adaptabilidade em arquiteturas de memória compartilhada. Naturalmente, com exceção de três delas, as soluções visam o multithreading, uma vez que o uso de threads é a forma trivial de explorar o paralelismo neste tipo de arquitetura. As estruturas geralmente permitem que as aplicações ajustem o número de threads para melhorar o uso de recursos, desempenho ou maior eficiência. A maioria das soluções é baseada em uma biblioteca e um Runtime/Middleware. A biblioteca fornece a interface para gerenciamento de encadeamentos e comunicação com o tempo de execução, que coleta informações sobre o uso de recursos ou execução de aplicações. Essas informações são usadas para determinar a alocação de recursos e o grau de paralelismo. Em geral, o uso desses mecanismos exige alguns ajustes no código-fonte original e, em alguns casos, nenhuma modificação é necessária, uma vez que a biblioteca modificada faz o trabalho pesado.

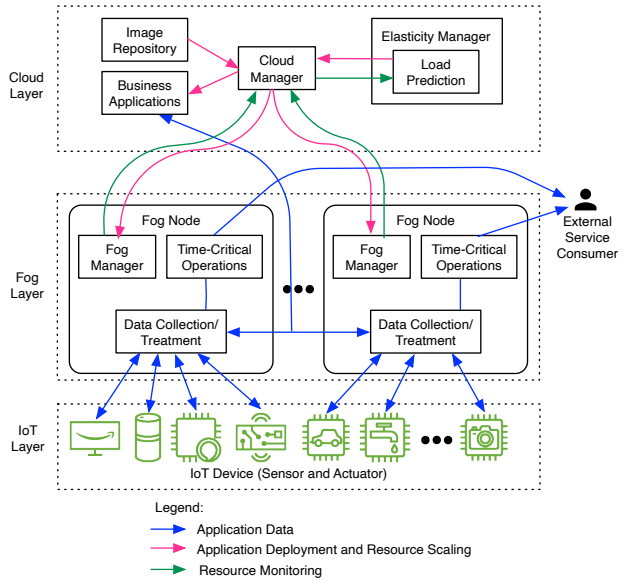


Figura 1.4. Arquitetura de fog baseada em elasticidade para tratar um alto volume de dados provenientes de dispositivos IoT.

Em relação aos esforços em prover adaptabilidade para clusters, pode-se observar um predomínio de soluções voltadas para MPI, uma vez que a maioria das aplicações que executam em clusters de computação de alto desempenho (HPC) são implementadas usando este modelo de programação. A execução de aplicações MPI dinâmicas depende do aumento ou diminuição do número de processos de acordo com as variações na disponibilidade dos recursos. As operações de maleabilidade são manipuladas por uma API que interage com um runtime/middleware para receber informações sobre a disponibilidade de recursos ou desempenho da aplicação. De acordo com essas informações, a API lança as ações de reconfiguração adequadas para adaptar a aplicação ao novo cenário. Como parte das operações crescentes, a API/runtime garante que novos processos sejam alocados nos recursos recém-alocados. Em operações de redução, as soluções devem liberar os recursos não utilizados.

Um desafio na reconfiguração dinâmica de aplicações MPI é escolher o número correto de processos/recursos a serem usados. As ações de reconfiguração só devem ser acionadas se a adição ou remoção do processo puder beneficiar o desempenho da aplicação. Para algumas classes de aplicações, aumentar o número de processadores além de um certo ponto pode não melhorar o tempo de execução (não escalável). Outro problema crítico é a necessidade de redistribuir os dados após a reconfiguração da aplicação. É fundamental, por exemplo, para aplicações fortemente acopladas, uma vez que o próprio processo apresenta altas demandas de comunicação, pode causar desbalanceamento de carga e modificar os padrões de comunicação.

No contexto da computação em grade, temos soluções que visam diferentes objetivos, como aumentar a robustez, garantir um determinado nível de QoS, melhorar o desempenho, fornecer tolerância a falhas e uso eficiente de recursos. Em termos de mecanismos usados, a maioria das soluções é baseada em técnicas de checkpoint/restart. As

aplicações são verificadas, reconfiguradas, reiniciadas e reescalonadas em um conjunto diferente de recursos. A principal vantagem do uso de checkpoints é que é uma técnica muito geral que pode ser aplicada a aplicações paralelas. As soluções propostas não são transparentes, exigindo algum esforço de programação, seja para adicionar anotações ou modificar o código-fonte para instrumentar as ações de adaptabilidade.

Ao abordar a adaptabilidade em ambientes de nuvem, observa-se que as estratégias de elasticidade de recursos e balanceamento de carga estão se integrando ao lidar com o desempenho. Em particular, o principal desafio aqui consiste em alterar o código da aplicação, que geralmente é escrito em MPI. O MPI foi proposto na década de 90 e hoje temos uma variedade de aplicações que usam essa API. A ideia de inserir elasticidade e chamadas de API dentro de um código MPI não é trivial, pois o desenvolvedor deve ter conhecimento profundo sobre a aplicação e das funcionalidades da nuvem. Por outro lado, observamos que cada vez mais iniciativas de HPC baseadas em nuvem estão explorando a elasticidade sem esforço (do ponto de vista do usuário). Aqui, o AutoElastic e o SelfElastic são sistemas que inserem elasticidade no momento da compilação, livrando o usuário de qualquer decisão sobre a reorganização de recursos e processos. Além disso, outro tópico importante é o comportamento da aplicação quando novos recursos estão disponíveis. Normalmente, temos a abordagem stop-reconfigure-and-go para usar mais recursos. Embora essa abordagem possa ser eficiente para as demandas corporativas, ela não se encaixa nas demandas de HPC. Nesses casos, a proposta de elasticidade assíncrona aparece como uma solução que orquestra o gerenciador de nuvem e a aplicação para evitar que a aplicação execute ações de elasticidade de recursos.

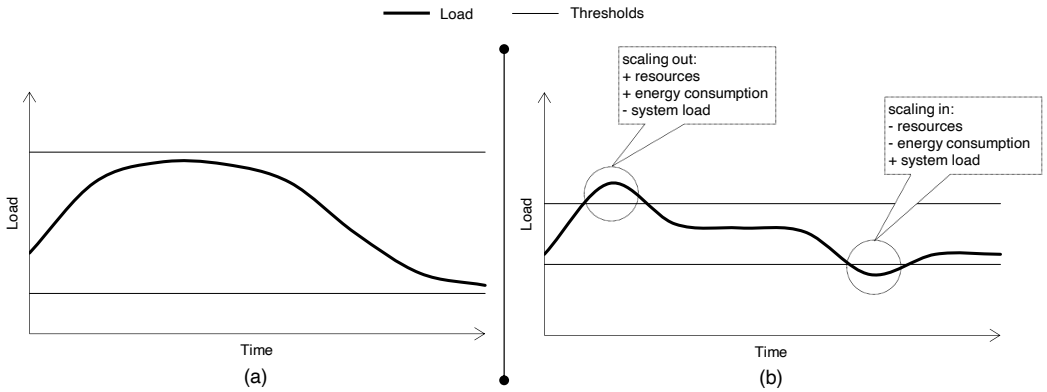
Por fim, no escopo da computação em névoa, percebe-se que a maioria dos trabalhos está focada nas demandas de IoT. Sensores e atuadores IoT são conectados aos recursos de névoa, que são então conectados à nuvem. As aplicações são então conectadas diretamente à névoa para obter resultados mais rapidamente e à nuvem para obter insights de IA mais robustos. A computação de alto desempenho é necessária para fornecer respostas oportunas às aplicações e atuadores IoT. As demandas vêm do número de sensores e atuadores, além do número de aplicações conectadas. Portanto, as preocupações com a escalabilidade são essenciais e percebe-se que a adaptação é tratada usando a elasticidade dos recursos nas camadas de névoa e nuvem para abordar essa temática. Além do desempenho, a elasticidade também é pertinente para reduzir os custos financeiros relacionados à manutenção de contêineres ou máquinas virtuais ligadas, já que seu número corresponde às demandas de um determinado momento.

1.3. Perspectivas e Tendências

Levando em consideração o atual estado da arte, apresenta-se a seguir algumas tendências futuras e oportunidades de pesquisa:

- Soluções leves baseadas em IA - Prevê-se o uso de mais e mais soluções de IA nas soluções de escalonamento e balanceamento de carga de forma que a previsão, o reconhecimento de padrões, a classificação instantânea e a correlação de eventos possam ajudar a otimizar as decisões de desempenho. O desafio aqui consiste em abordar procedimentos demorados relacionados à IA. Assim, é possível usar a ideia de treinar uma vez, usar várias vezes e a combinação de IA com abordagens heurísticas

rápidas. Os artigos recentes publicados por Jiang et al. [Jiang et al. 2021] e Yadav et al. [Yadav et al. 2021] confirmam essa tendência, em que técnicas de aprendizado de máquina são usadas para fornecer provisionamento de recursos eficiente. A Figura 1.5 mostra um exemplo em que podemos usar soluções de IA para combinar vários objetivos na execução de aplicações HPC em arquiteturas de névoa e nuvem.



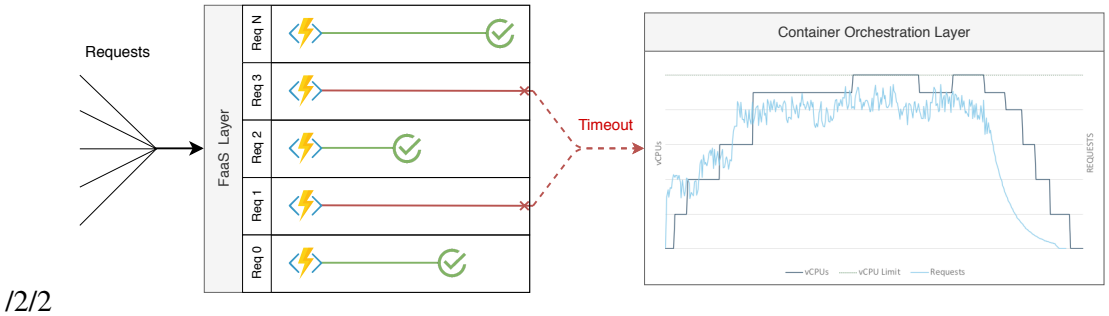
/2/2

Figura 1.5. Possibilidade de usar soluções baseadas em IA para melhorar não só o desempenho, mas também a energia e o orçamento na execução de aplicações HPC na nuvem. Em (a), temos uma aplicação que executa dentro de limites predefinidos. Em (b), temos o controle baseado em IA para manter a aplicação dentro dos limites levando em consideração diversos critérios objetivos.

- Combinação de metodologias de elasticidade de nuvem - É possível combinar elasticidade vertical e horizontal em arquiteturas de névoa e nuvem. Primeiramente, a vertical pode ser usada até o limite de uma máquina física. Se mais desempenho for realmente necessário, podemos usar a elasticidade horizontal para alocar mais recursos virtuais. Essa combinação pode ser interessante para o consumo de energia e redução de custos financeiros, uma vez que primeiro aloca-se as demandas a uma máquina física específica, usando-a de forma eficiente antes de alocar outras. Além disso, a combinação de Serverless ou FaaS (Function as a Service) e as abordagens tradicionais de elasticidade da nuvem podem ser interessantes para executar aplicações irregulares, uma vez que FaaS é mais adequado para tarefas de execução curta. Ao mesmo tempo, a alocação de servidores pode lidar com demandas de longa duração de uma maneira melhor, conforme ilustrado na Figura 1.6.

- Uso eficiente de arquiteturas heterogêneas - Com tecnologias de aceleradores (por exemplo, GPU e FPGA), as arquiteturas de sistema entraram em uma tendência clara para o aumento do paralelismo e da heterogeneidade. No entanto, as estruturas de alocação de recursos que visam arquiteturas heterogêneas não atingiram seu potencial total, que ocorre ao orquestrar todos os diferentes recursos juntos ou dinamicamente, selecionando a configuração de recurso mais adequada para cada aplicação (ou estágio da aplicação).

- Linguagens de programação promissoras - É previsto o uso de linguagens de programação como Go e Elixir/Erlang como uma tendência HPC. Essas linguagens possuem recursos integrados para lidar com computação de alto desempenho, incluindo



/2/2

Figura 1.6. Combinação de Function as a Service (FaaS) e elasticidade tradicional reativa para executar aplicações HPC.

diretivas de exclusão mútua, mecanismos eficientes de envio/recebimento de dados, replicação de dados, suporte de escalabilidade e compatibilidade de arquiteturas baseadas em memória distribuída e compartilhada. Além disso, observamos o uso crescente dessas linguagens de programação com ferramentas de orquestração de contêineres, como K3S e Kubernetes, para lidar com propostas eficientes de programação e elasticidade.

- Exploração da computação de alto desempenho de forma automática - Como tendência, observa-se a exploração de abordagens automáticas para impulsionar aplicações HPC já desenvolvidas a executar ainda mais rápido. A ideia aqui é explorar bibliotecas de programação e compiladores que alteram o código de funções para inserir transparentemente nas diretivas de gerenciamento de recursos do ponto de vista do usuário. Assim, é possível transformar uma aplicação não elástica MPI em uma elástica, apenas computando-a com uma biblioteca baseada em elasticidade MPI particular, permitindo que um código explore os benefícios da computação em nuvem.

- Mecanismos de redistribuição de dados - O desafio de projetar técnicas de adaptabilidade para algumas classes de aplicações (por exemplo, SPMD e decomposição de domínio) não é simplesmente modificar o número de processos em que a aplicação está rodando de acordo com a disponibilidade de recursos. As ações de reconfiguração envolvem redistribuir os dados pelos novos processos (o que pode causar desbalanceamento de carga) e modificar os padrões de comunicação. Nesse sentido, mecanismos de redistribuição de dados transparentes para o usuário são necessários para permitir o uso eficiente de recursos dinâmicos por uma ampla classe de aplicações.

1.4. Conclusão

Este minicurso apresentou como o estado da arte está tratando da adaptabilidade em aplicações paralelos nas últimas duas décadas. Para os próximos anos, prevê-se o uso crescente da computação em nuvem para executar demandas de HPC, permitindo o uso de elasticidade e a alocação de hardware específico, incluindo requisitos de cache e dispositivos de GPU, definições de QoS e especificações de bibliotecas e dependências. Além

disso, ressalta-se dois tópicos importantes para os próximos anos. Em primeiro lugar, a exploração da elasticidade sem esforço, automática e transparente são os principais motivadores para explorar HPC em arquiteturas virtualizadas, permitindo que os desenvolvedores executem suas demandas mais rapidamente com mudanças mínimas de código. Em segundo lugar, alinhado às tendências do Top500.org, se tornará necessário cada vez mais que programas extraiam o poder simultâneo de múltiplas arquiteturas. Assim, novos middleware e bibliotecas devem permitir que um código possa ser executado, de forma automática, em uma gama de arquiteturas (multicomputadores, multiprocessadores e aceleradores, por exemplo) com o mínimo de esforço.

Referências

[Al-khafajiy et al. 2019] Al-khafajiy, M., Baker, T., Al-Libawy, H., Maamar, Z., Aloqaily, M., and Jararweh, Y. (2019). Improving fog computing performance via fog-2-fog collaboration. *Future Generation Computer Systems*, 100:266 – 280.

[Aldinucci et al. 2006] Aldinucci, M., Coppola, M., Danelutto, M., Tonellotto, N., Vanneschi, M., and Zoccolo, C. (2006). High level grid programming with ASSIST. *Computational Methods in Science and Technology*, 12(1):21–32.

[Batheja and Parashar 2003] Batheja, J. and Parashar, M. (2003). A framework for adaptive cluster computing using javaspaces. *Cluster Computing*, 6(3):201–213.

[Buisson et al. 2007] Buisson, J., Andre, F., and Pazat, J.-L. (2007). Supporting adaptable applications in grid resource management systems. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, page 58–65, USA. IEEE Computer Society.

[Catalán et al. 2019] Catalán, S., Herrero, J. R., Quintana-Ortí, E. S., Rodríguez-Sánchez, R., and Van De Geijn, R. (2019). A case for malleable thread-level linear algebra libraries: The lu factorization with partial pivoting. *IEEE Access*, 7:17617–17633.

[Cera 2011] Cera, M. C. (2011). *Providing adaptability to MPI applications on current parallel architectures*. PhD thesis, Universidade Federal do Rio Grande do Sul. Instituto de Informática.Programa de Pós-Graduação em Computação.

[Chen et al. 2017] Chen, Y., Chang, Y., Chen, C., Lin, Y., Chen, J., and Chang, Y. (2017). Cloud-fog computing for information-centric internet-of-things applications. In *2017 International Conference on Applied System Innovation (ICASI)*, pages 637–640.

[Cho et al. 2018] Cho, Y., Guzman, C. A. C., and Egger, B. (2018). Maximizing system utilization via parallelism management for co-located parallel applications. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, PACT '18, pages 1–14, New York, NY, USA. Association for Computing Machinery.

[Choi et al. 2016] Choi, Y., Alsaffar, A. A., , Pham, H. P., Hong, C., Huh, E., and Aazam, M. (2016). An Architecture of IoT Service Delegation and Resource Allocation Based

on Collaboration between Fog and Cloud Computing. *Mobile Information Systems*, 2016.

[Comprés et al. 2016] Comprés, I., Mo-Hellenbrand, A., Gerndt, M., and Bungartz, H.-J. (2016). Infrastructure and api extensions for elastic execution of mpi applications. In *Proceedings of the 23rd European MPI Users' Group Meeting*, EuroMPI 2016, page 82–97, New York, NY, USA. Association for Computing Machinery.

[Creech 2015] Creech, T. M. (2015). Efficient multiprogramming for multicores with scaf. Master's thesis, Faculty of the Graduate School of the University of Maryland.

[da Rosa Righi et al. 2016] da Rosa Righi, R., Rodrigues, V. F., da Costa, C. A., Galante, G., Bona, L. C. E. D., and Ferreto, T. C. (2016). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *IEEE Trans. Cloud Comput.*, 4(1):6–19.

[D'Amico et al. 2018] D'Amico, M., Garcia-Gasulla, M., López, V., Jokanovic, A., Sirvent, R., and Corbalan, J. (2018). Drom: Enabling efficient and effortless malleability for resource managers. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, ICPP '18, pages 1–10, New York, NY, USA. Association for Computing Machinery.

[Dominico et al. 2018] Dominico, S., de Almeida, E. C., Meira, J. A., and Alves, M. A. (2018). An elastic multi-core allocation mechanism for database systems. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 473–484.

[Dongarra et al. 2011] Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Chi, X., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Jin, Z., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M. S., Nagel, W. E., Nakashima, H., Papka, M. E., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., Van Der Steen, A., Vetter, J., Williams, P., Wisniewski, R., and Yelick, K. (2011). The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60.

[El Maghraoui et al. 2009] El Maghraoui, K., Desell, T. J., Szymanski, B. K., and Varela, C. A. (2009). Malleable iterative mpi applications. *Concurrency and Computation: Practice and Experience*, 21(3):393–413.

[Feitelson and Rudolph 1996] 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*, volume 1162 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag.

[Foster et al. 2008] Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. IEEE.

[Fox et al. 2017] Fox, W., Ghoshal, D., Souza, A., Rodrigo, G. P., and Ramakrishnan, L. (2017). E-hpc: A library for elastic resource management in hpc environments. In *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science*, WORKS '17, pages 1–11, New York, NY, USA. Association for Computing Machinery.

[Galante and Bona 2012] Galante, G. and Bona, L. C. E. (2012). A survey on cloud computing elasticity. In *Proceedings of the International Workshop on Clouds and eScience Applications Management*, CloudAM'12, pages 263–270. IEEE.

[Galante and Bona 2014] Galante, G. and Bona, L. C. E. (2014). Supporting elasticity in openmp applications. In *Proceedings of the 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '14, page 188–195, USA. IEEE Computer Society.

[Galante and da Rosa Righi 2017] Galante, G. and da Rosa Righi, R. (2017). Exploring cloud elasticity in scientific applications. In Antonopoulos, N. and Gillam, L., editors, *Cloud Computing - Principles, Systems and Applications, Second Edition*, Computer Communications and Networks, pages 101–125. Springer.

[Galante and Erpen De Bona 2015] Galante, G. and Erpen De Bona, L. C. (2015). A programming-level approach for elasticizing parallel scientific applications. *Journal of Systems and Software*, 110:239–252.

[Georgakoudis et al. 2017] Georgakoudis, G., Vandierendonck, H., Thoman, P., Supinski, B. R. D., Fahringer, T., and Nikolopoulos, D. S. (2017). Scalo: Scalability-aware parallelism orchestration for multi-threaded workloads. *ACM Trans. Archit. Code Optim.*, 14(4).

[Gordon and Lu 2011] Gordon, A. W. and Lu, P. (2011). Elastic phoenix: Malleable mapreduce for shared-memory systems. In Altman, E. R. and Shi, W., editors, *Network and Parallel Computing - 8th IFIP International Conference, NPC 2011*, volume 6985 of *Lecture Notes in Computer Science*, pages 1–16. Springer.

[Grelck 2015] Grelck, C. (2015). Moldable applications on multi-core servers: Active resource management instead of passive resource administration. In *Proceedings of the 18. Kolloquium Programmiersprachen und Grundlagen der Programmierung*, KPS 2015, pages 1–10. TU Wien.

[Gupta et al. 2014] Gupta, A., Acun, B., Sarood, O., and Kalé, L. V. (2014). Towards realizing the potential of malleable jobs. In *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–10.

[He et al. 2018] He, J., Wei, J., Chen, K., Tang, Z., Zhou, Y., and Zhang, Y. (2018). Multitier fog computing with large-scale iot data analytics for smart cities. *IEEE Internet of Things Journal*, 5(2):677–686.

[Herbst et al. 2013] Herbst, N. R., Kounev, S., and Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing*, ICAC'13, pages 23–27. USENIX.

[Huang et al. 2004] Huang, C., Lawlor, O., and Kalé, L. V. (2004). Adaptive mpi. In Rauchwerger, L., editor, *Languages and Compilers for Parallel Computing*, pages 306–322, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Hungershöfer and Wierum 2002] Hungershöfer, J. and Wierum, J. (2002). On the quality of partitions based on space-filling curves. In Sloot, P. M. A., Tan, C. J. K., Dongarra, J. J., and Hoekstra, A. G., editors, *Computational Science - ICCS 2002, International Conference, Amsterdam, The Netherlands, April 21-24, 2002. Proceedings, Part III*, volume 2331 of *Lecture Notes in Computer Science*, pages 36–45. Springer.

[Iserte et al. 2018] Iserte, S., Mayo, R., Quintana-Ortí, E. S., Beltran, V., and Peña, A. J. (2018). Dmr api: Improving cluster productivity by turning applications into malleable. *Parallel Computing*, 78:54–66.

[Iserte and Rojek 2020] Iserte, S. and Rojek, K. (2020). An study of the effect of process malleability in the energy efficiency on gpu-based clusters. *J. Supercomput.*, 76(1):255–274.

[Jiang et al. 2021] Jiang, Y., Kodialam, M., Lakshman, T. V., Mukherjee, S., and Tassiulas, L. (2021). Resource allocation in data centers using fast reinforcement learning algorithms. *IEEE Transactions on Network and Service Management*.

[Kalé et al. 2002] Kalé, L. V., Kumar, S., and DeSouza, J. (2002). A malleable-job system for timeshared parallel machines. In *Proceedings of the 2Nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 230–, Washington, DC, USA. IEEE Computer Society.

[Kale 2020] Kale, V. (2020). *Parallel computing architectures and APIs : IoT big data stream processing*. CRC Press, Taylor & Francis Group, Boca Raton, FL.

[Kehrer and Blochinger 2020] Kehrer, S. and Blochinger, W. (2020). Equilibrium: an elasticity controller for parallel tree search in the cloud. *J. Supercomput.*, 76(11):9211–9245.

[Kennedy et al. 2002] Kennedy, K., Mazina, M., Mellor-Crummey, J. M., Cooper, K. D., Torczon, L., Berman, F., Chien, A. A., Dail, H., Sievert, O., Angulo, D., Foster, I. T., Aydt, R. A., Reed, D. A., Gannon, D., Johnsson, S. L., Kesselman, C., Dongarra, J., Vadhiyar, S. S., and Wolski, R. (2002). Toward a framework for preparing and executing adaptive grid programs. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, page 322, USA. IEEE Computer Society.

[Kim et al. 2011] Kim, D., Larson, J. W., and Chiu, K. (2011). Toward malleable model coupling. *Procedia Computer Science*, 4:312–321. Proceedings of the International Conference on Computational Science, ICCS 2011.

[Klein and Perez 2011] Klein, C. and Perez, C. (2011). An rms for non-predictably evolving applications. In *2011 IEEE International Conference on Cluster Computing*, pages 326–334.

[Klemm et al. 2009] Klemm, M., Bezold, M., Gabriel, S., Veldema, R., and Philippsen, M. (2009). Reparallelization techniques for migrating openmp codes in computational grids. *Concurrency and Computation: Practice and Experience*, 21(3):281–299.

[Lemarinier et al. 2016] Lemarinier, P., Hasanov, K., Venugopal, S., and Katrinis, K. (2016). Architecting malleable mpi applications for priority-driven adaptive scheduling. In *Proceedings of the 23rd European MPI Users' Group Meeting*, EuroMPI 2016, page 74–81, New York, NY, USA. Association for Computing Machinery.

[Leopold et al. 2006] Leopold, C., Süß, M., and Breitbart, J. (2006). Programming for malleability with hybrid mpi-2 and openmp: Experiences with a simulation program for global water prognosis. In *Proceedings of the European Conference on Modelling and Simulation*, pages 665–670.

[Libutti et al. 2020] Libutti, L. A., Igual, F. D., Piñuel, L., De Giusti, L., and Naiouf, M. (2020). Towards a malleable tensorflow implementation. In Rucci, E., Naiouf, M., Chichizola, F., and De Giusti, L., editors, *Cloud Computing, Big Data & Emerging Topics*, pages 30–40, Cham. Springer International Publishing.

[Liu and Weissman 2015] Liu, F. and Weissman, J. B. (2015). Elastic job bundling: An adaptive resource request strategy for large-scale parallel applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 1–12, New York, NY, USA. Association for Computing Machinery.

[Martín et al. 2015] Martín, G., Singh, D. E., Marinescu, M.-C., and Carretero, J. (2015). Enhancing the performance of malleable mpi applications by using performance-aware dynamic reconfiguration. *Parallel Comput.*, 46(C):60–77.

[Mascagni et al. 2019] Mascagni, M., Iserte, S., Martínez, H., Barrachina, S., Castillo, M., Mayo, R., and Peña, A. J. (2019). Dynamic reconfiguration of noniterative scientific applications: A case study with hpg aligner. *Int. J. High Perform. Comput. Appl.*, 33(5):804–816.

[Mayes et al. 2005] Mayes, K., Luján, M., Riley, G., Chin, J., Coveney, P., and Gurd, J. (2005). Towards performance control on the grid. *Phil. Trans. R. Soc.*, 363(1833):1793–1805.

[McFarland 2011] McFarland, D. J. (2011). Exploiting malleable parallelism on multicore systems. Master's thesis, Faculty of the Virginia Polytechnic Institute and State University.

[Mo-Hellenbrand et al. 2017] Mo-Hellenbrand, A., Comprés, I., Meister, O., Bungartz, H.-J., Gerndt, M., and Bader, M. (2017). A large-scale malleable tsunami simulation realized on an elastic mpi infrastructure. In *Proceedings of the Computing Frontiers Conference*, CF'17, page 271–274, New York, NY, USA. Association for Computing Machinery.

[Naha et al. 2020] Naha, R. K., Garg, S., Chan, A., and Battula, S. K. (2020). Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. *Future Generation Computer Systems*, 104:131 – 141.

[Nguyen et al. 2020] Nguyen, N. D., Phan, L. A., Park, D. H., Kim, S., and Kim, T. (2020). Elasticfog: Elastic resource provisioning in container-based fog computing. *IEEE Access*, 8:183879–183890.

[Pagani et al. 2016] Pagani, D. H., Bona, L. C. E. D., and Galante, G. (2016). Uma abordagem baseada em níveis de estresse para alocação elástica de recursos em sistema de bancos de dados. In *Anais do XIV Workshop em Clouds e Aplicações*, WCGA 2016, pages 1–14, Porto Alegre. SBC.

[Prabhakaran et al. 2014] Prabhakaran, S., Iqbal, M., Rinke, S., Windisch, C., and Wolf, F. (2014). A batch system with fair scheduling for evolving applications. In *Proceedings of the 2014 Brazilian Conference on Intelligent Systems*, BRACIS '14, page 351–360, USA. IEEE Computer Society.

[Rajan and Thain 2017] Rajan, D. and Thain, D. (2017). Designing self-tuning split-map-merge applications for high cost-efficiency in the cloud. *IEEE Transactions on Cloud Computing*, 5(2):303–316.

[Raveendran et al. 2011] Raveendran, A., Bicer, T., and Agrawal, G. (2011). A framework for elastic execution of existing mpi programs. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 940–947.

[Ribeiro et al. 2013] Ribeiro, F., Rebello, V., Nascimento, A., Boeres, C., and Sena, A. (2013). Autonomic malleability in iterative mpi applications. In *Proceedings of the 2013 25th International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '13, page 192–199, USA. IEEE Computer Society.

[Rodrigues et al. 2018] Rodrigues, V. F., da Rosa Righi, R., da Costa, C. A., Singh, D., Muñoz, V. M., and Chang, V. (2018). Towards combining reactive and proactive cloud elasticity on running HPC applications. In Muñoz, V. M., Wills, G. B., Walters, R. J., Firouzi, F., and Chang, V., editors, *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security, IoTBDS 2018*, pages 261–268. SciTePress.

[Rodrigues et al. 2017] Rodrigues, V. F., da Rosa Righi, R., Rostirolla, G., Barbosa, J. L. V., da Costa, C. A., Alberti, A. M., and Chang, V. I. (2017). Towards enabling live thresholding as utility to manage elastic master-slave applications in the cloud. *J. Grid Comput.*, 15(4):535–556.

[Small et al. 2017] Small, N., Akkermans, S., Joosen, W., and Hughes, D. (2017). Niflheim: An end-to-end middleware for applications on a multi-tier iot infrastructure. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pages 1–8.

[Spenke et al. 2019] Spenke, F., Balzer, K., Frick, S., Hartke, B., and Dieterich, J. M. (2019). Malleable parallelism with minimal effort for maximal throughput and maximal hardware load. *Computational and Theoretical Chemistry*, 1151:72–77.

[Stallings 2017] Stallings, W. (2017.). *Computer organization and architecture*. Pearson Education, Inc., Hoboken, New Jersey, 10th ed. edition.

[Sudarsan and Ribbens 2007] Sudarsan, R. and Ribbens, C. J. (2007). Reshape: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, page 44, USA. IEEE Computer Society.

[Sudarsan et al. 2009] Sudarsan, R., Ribbens, C. J., and Farkas, D. (2009). Dynamic resizing of parallel scientific simulations: A case study using lammps. In *Proceedings of the 9th International Conference on Computational Science: Part I*, ICCS '09, page 175–184, Berlin, Heidelberg. Springer-Verlag.

[Suleman et al. 2008] Suleman, M. A., Qureshi, M. K., and Patt, Y. N. (2008). Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps. *SIGARCH Comput. Archit. News*, 36(1):277–286.

[Utrera et al. 2004] Utrera, G., Corbalan, J., and Labarta, J. (2004). Implementing malleability on mpi jobs. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, PACT '04, page 215–224, USA. IEEE Computer Society.

[Vadhiyar and Dongarra 2003] Vadhiyar, S. S. and Dongarra, J. J. (2003). Srs: A framework for developing malleable and migratable parallel applications for distributed systems. *Parallel Processing Letters*, 13(02):291–312.

[Van Nieuwpoort et al. 2010] Van Nieuwpoort, R. V., Wrzesińska, G., Jacobs, C. J. H., and Bal, H. E. (2010). Satin: A high-level and efficient grid programming model. *ACM Trans. Program. Lang. Syst.*, 32(3).

[Wilkinson 2009] Wilkinson, B. (2009). *Grid Computing: Techniques and Applications*. CRC Press, Boca Raton, FL, 1st ed. edition.

[Wrzesinska et al. 2005] Wrzesinska, G., van Nieuwpoort, R., Maassen, J., and Bal, H. (2005). Fault-tolerance, malleability and migration for divide-and-conquer applications on the grid. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 10 pp.–.

[Yadav et al. 2021] Yadav, M. P., Rohit, and Yadav, D. K. (2021). Resource provisioning through machine learning in cloud services. *Arabian Journal for Science and Engineering*.

[Yin et al. 2018] Yin, L., Luo, J., and Luo, H. (2018). Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. *IEEE Transactions on Industrial Informatics*, 14(10):4712–4721.