Mapeamento Sistemático de Ferramentas que Auxiliam na Instrumentação de *Checkpoints* em Aplicações de Computação de Alto Desempenho

Andrei dos Santos Mattos[1], **Rodrigo Porfírio da Silva Sacchi**[1]

[1]Faculdade de Ciências Exatas e Tecnologia
Universidade Federal da Grande Dourados (UFGD)

andreimattos06@gmail.com, rodrigosacchi@ufgd.edu.br

Resumo. *Ferramentas de checkpointing são usadas para salvar o estado de uma aplicação, seja ela sequencial ou distribuída. Em anos recentes, surgiram diversas ferramentas de checkpointing como alternativas para efetuar checkpoints locais ou globais, este último principalmente no caso de aplicações de computação de alto desempenho. Entretanto, não se tem um conhecimento global sobre e como essas ferramentas podem ser utilizadas em aplicações, se elas fazem uso de checkpoint local ou global, em qual nível, usuário, aplicação ou kernel a aplicação trabalha, se a aplicação fornece um meio automático de instrumentar checkpoints ou ainda se o algoritmo de checkpoint que a ferramenta implementa é síncrono, assíncrono ou quase-síncrono. Desta perspectiva, este trabalho tem por objetivo o desenvolvimento de uma mapeamento sistemático sobre essas ferramentas, de modo a esclarecer esses pontos de interesse.*

Abstract. *Checkpointing tools are used to save the state of an application, be it sequential or distributed. In recently years, emerged various checkpointing tools as alternatives to do local or global checkpoints, the last one specially in case of high performance computation. However, there isn't global knowledge about and how this tools can be used in applications, if they make use of local or global checkpoint, in which level, user, application or kernel the application works, if the application provides an automatic way to instrument checkpoints or even if the checkpoint algorithm that the tool implements is synchronous, assynchronous, almost syncrhonous. From this perspective, this work have as the objective the development of a systematic mapping about this tools, in order to clarify this points of interest.*

1. Introdução

O Top500[1] fornece uma lista com estatísticas sobre os 500 computadores mais poderosos da atualidade, que é atualizada duas vezes por ano. A última lista gerada, de Novembro de 2018, conta com cinco supercomputadores do Departamento de Energia (DOE) dos EUA entre os 10 primeiros colocados, sendo os dois primeiros o Summit, do Oak Ridge National Laboratory (ORNL) e a Sierra, do Lawrence Livermore National Laboratory (LLNL).

[1]https://www.top500.org

É fácil observar nesta lista que o número de nós de computação aumenta frequentemente. Contudo, este aumento traz como desvantagem o crescimento da taxa de falhas. De acordo com [Cappello et al. 2014], o Tempo Médio Entre Falhas (*MTBF – Mean Time Between Failure*) tem reduzido de dias para algumas horas. Aplicações de computação de alto desempenho requerem um longo tempo para concluir suas execuções, que podem demorar de algumas horas ou até dias. Contudo, se o *MTBF* do *cluster* é de apenas algumas horas, sua aplicação falhará. Neste sentido, há necessidade de usar um mecanismo de tolerância à falhas. De acordo com [Snir et al. 2014], as técnicas de tolerância à falhas são:

- Detecção de erros, que identifica a presença de erros;
- Recuperação, que evita falhas.

Uma das técnicas que mais tem sido estudadas é a recuperação por retrocesso (*rollback*), que tem a finalidade de reverter a execução da aplicação a um estado correto quando ocorrem falhas. Para recuperar uma aplicação de computação de alto desempenho, é necessário salvar, periodicamente, o estado da aplicação. Esses estados salvos são denominados *checkpoints* ou pontos de recuperação. O sentido é que, após uma falha, sua aplicação retroceda e reinicie a partir de um estado salvo.

Na computação de alto desempenho, a principal dificuldade no uso de mecanismos de recuperação por retrocesso baseados em *checkpoints* é determinar um ponto de recuperação global. Neste sentido, diversos protocolos de *checkpointing* foram propostos nos últimos 25 anos [Baldoni et al. 1997, Cao and Singhal 2003, Garcia et al. 2001, Garcia and Buzzato 2001, Garg et al. 2010, Hélary et al. 2000, Kshemkalyani 2009, Kshemkalyani 2010, Luo and Manivannan 2009, Manivannan and Singhal 1996, Vieira et al. 2001, Wang 1997, Tsai and Lin 2005], entre outros. Além disso, muitas ferramentas foram desenvolvidas para permitir que os estados globais de aplicações de computação de alto desempenho sejam salvos em um meio de armazenamento estável. Dentre estas ferramentas, é possível citar [Ansel et al. 2009, Hursey et al. 2007, Rodríguez et al. 2010, Ruscio et al. 2007], entre outras.

Com o intuito de fazer um estudo sobre estas ferramentas desenvolvidas e descritas na literatura, este trabalho propõe um mapeamento sistemático sobre ferramentas que auxiliam na instrumentação de *checkpoints* em aplicações de computação de alto desempenho. De acordo com [Petersen et al. 2008], um mapeamento sistemático é uma metodologia que envolve a busca na literatura para averiguar a natureza, a extensão e a qualidade de estudos publicados dentro de uma determinada área.

Para o desenvolvimento deste trabalho, usou-se como guia a metodologia apresentada por [Petersen et al. 2015]. O restante deste trabalho está organizado da seguinte forma: A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve a metodologia de pesquisa, apresentando as questões de pesquisa, *string* de busca, critérios de inclusão e exclusão, as bases de dados, entre outras informações relevantes. A Seção 4 apresenta os resultados do trabalho, respondendo as questões de pesquisa. Finalmente, a Seção 5 apresenta as considerações finais sobre este trabalho.

2. Trabalhos Relacionados

Diversas pesquisas fazem uso do mapeamento sistemático para alcançar uma visão geral sobre uma área de pesquisa ou para dar um rumo para alguma pesquisa em seu início. Contudo, na área de computação distribuída não se publica estudos relacionados a mapeamentos sistemáticos. [Elnozahy et al. 2002], por exemplo, apresenta um *survey* sobre protocolos que usam a recuperação por retrocesso em bibliotecas de trocas de mensagens, classificam esses protocolos em diferentes categorias e também apresentam questões de pesquisa centrais para a recuperação por retrocesso. [Kalaiselvi and Rajaraman 2000] examina algoritmos que foram apresentados na literatura sobre *checkpointing* em sistemas paralelos/distribuídos. Eles observaram que a maioria dos algoritmos publicados para verificação em sistemas de transmissão de mensagens é baseada no algoritmo clássico de [Chandy and Lamport 1985].

Outros trabalhos, como [Gärtner 1999, Egwutuoha et al. 2013, Treaster 2005] apresentam artigos que examinam mecanismos de tolerância à falhas. [Shahzad et al. 2013] examina um conjunto de técnicas de recuperação por retrocesso com o uso de *checkpoints*. No trabalho, os autores implementaram diferentes categorias de algoritmos e os compararam através do *overhead* de *checkpointing*. Contudo, nenhum deles faz um mapeamento sistemático sobre ferramentas que executam *checkpointing* em aplicações distribuídas.

3. Método de Pesquisa

O método de pesquisa utilizado neste trabalho é o **mapeamento sistemático** o qual, segundo [Petersen et al. 2008], é projetado para dar uma visão geral de uma área de pesquisa através de classificação e contando as contribuições em relação às categorias dessa classificação. Também envolve pesquisar, na literatura, a fim de saber quais tópicos foram abordados nos trabalhos mapeados e onde estes foram publicados.

[Petersen et al. 2008] determina as etapas essenciais para a realização de um mapeamento sistemático, sendo elas:

1. Definição de questões de pesquisa;
2. Realização da pesquisa de estudos primários relevantes;
3. Triagem dos documentos;
4. *Keywording* dos resumos;
5. Extração de dados e mapeamento;

Seguindo essas etapas é possível realizar um mapeamento sistemático satisfatório, porém, para cada trabalho existe a necessidade de adaptar uma ou mais etapas para a realidade do tema, para que não haja uma grande generalidade entre trabalhos mesmo que de áreas completamente diferentes.

3.1. Definição das questões de pesquisa

Nesta etapa foi estabelecido o corpo de todo o trabalho, foram estabelecidas as questões de pesquisas as quais foram cuidadosamente elaboradas para atenderem de forma abrangente o tema e identificar os tópicos abordados pelos trabalho, são elas:

1. QP_1 - A ferramenta realiza os *checkpoints* de maneira local ou global?

2. QP_2 - A ferramenta é executada em qual nível: *user-level*, *kernel-level* ou *application-level*?
3. QP_3 - O *checkpoint* distribuído é feito manualmente ou automaticamente?
4. QP_4 - O *checkpointing* é síncrono, assíncrono ou quase-síncrono?

Também foi definido quais seriam as bases de dados utilizadas para efetuar a busca dos trabalhos, as quais são: *IEEE Explore*, *Elsevier (Science Direct)* e *ACM Digital Library*. Essas bases de dados foram utilizadas pelo fato de existirem poucos trabalhos relacionados na literatura nacional, sendo assim as bases nacionais foram desconsideradas, desta forma foram selecionadas as três principais bases em relação a jornais, conferências e simpósios e também a base Scopus foi retirada pelo fato de haver muitos trabalhos repetidos nesta base. Para o processo de elaboração das strings de busca foi utilizado um processo muito parecido com o utilizado por [Petersen et al. 2015] onde através das questões de pesquisa foi possível selecionar palavras chaves e agrupá-las em conjuntos sendo o primeiro grupo composto pelas palavras *distributed* e *transparent*, devido a essas palavras estarem relacionadas as principais formas de *checkpoint*, em seguida o próximo conjunto é composto pelas palavras *framework*, *package*, *tool* e *architecture* pois identificam do que se trata a implementação e por fim a palavra *MPI* a qual incluiu todas as variações da palavra *MPI*.

De acordo com a base de dados foi elaborada uma *string* de busca, a qual antes de chegar em sua forma definitiva passou por vários testes, tendo em vista que os resultados obtidos pelas buscas mudavam drasticamente a medida que a *string* de busca era alterada, ocorrendo até mesmo situações em que uma pequena mudança na *string* não retornava nenhum resultado. A Tabela 1 apresenta a relação entre a *string* de busca para cada uma das bases utilizadas.

Tabela 1. Strings de busca utilizadas.

Base de Dados	***String* de Busca**
IEEE Explore	(distributed or transparent) and checkpoint* and (framework or package or tool or architecture) and (*mpi*)
ACM Digital Library	(distributed or transparent) and checkpoint* and (framework or package or tool or architecture) and (*mpi*)
Science Direct	(distributed or transparent) checkpoint* and (((((framework) or package) or tool) or architecture)) and (*mpi*))

3.2. Realização da pesquisa de estudos primários relevantes

Mesmo com a elaboração de uma *string* de busca adequada, ainda assim os resultados retornados da busca nem sempre atendem as necessidades do trabalho, sendo então necessário a realização de um processo de filtragem para que ao fim do processo restem

apenas trabalhos pertinentes ao mapeamento. Dessa forma foi definido os critérios de exclusão e inclusão para parametrizar a seleção dos trabalhos.

- Critérios de Inclusão:
  - Trabalho deve ser na área de computação distribuída.
  - Apenas trabalhos completos e disponíveis para download.
  - Ferramentas de *checkpoint* distribuído.
- Critérios de Exclusão:
  - Resumos, editorais, entre outros.
  - Apenas trabalhos revisados.
  - Trabalhos duplicados, nesse caso será considerado o trabalho que descreve melhor a ferramenta.

Por fim, para que os trabalhos encontrados na busca fossem de maneira geral trabalhos mais recentes, foi determinado que apenas trabalhos datados entre o ano de 2000 a 2019 são considerados.

3.3. Ameaças à Validade

Para proporcionar um processo de seleção justo e imparcial, os critérios de inclusão e exclusão juntamente com as questões de pesquisa foram determinados antes do início deste mapeamento sistemático.

Para a realização das triagens, foi considerado como dupla de pesquisadores o orientando e o orientador, que avaliaram cada trabalho obtido na etapa inicial, realizando as etapas subsequentes do processo e, em caso de discordância entre os avaliadores, foram realizadas discussões e leituras em conjunto com o intuito de resolver a divergência encontrada.

Como as buscas pelos trabalhos primários foram realizadas usando diferentes bases de dados internacionais, optou-se pelo uso de diferentes *strings* de busca, adaptadas à cada base.

4. Resultados

Através da execução do método de pesquisa descrito, a Tabela 2 resume, para cada base de dados considerada, as três etapas para a seleção de trabalhos usados no mapeamento sistemático.

A primeira etapa consistiu na realização das buscas nas bases de dados através do uso da *string* de busca, onde foram retornados os trabalhos que o motor de busca da base de dados identificou, totalizando 184 trabalhos somadas as três bases de dados.

Tabela 2. Etapas do mapeamento sistemástico.

	Quantidade de Trabalhos		
Base de Dados	**Primeira Etapa**	**Segunda Etapa**	**Terceira Etapa**
IEEE Xplore	71	12	10
ACM Digital Library	61	7	7
Elsevier (Science Direct)	52	3	2
Total	184	22	19

Na segunda etapa foi realizada a leitura dos títulos, resumos e palavras-chaves de todos os trabalhos retornados na primeira etapa. De acordo com a leitura, caso fosse identificado que o trabalho seria relevante para o mapeamento sistemático, ele seria selecionado, caso contrário ele seria descartado. É notável que houve uma diminuição drástica entre os números obtidos entre essas duas primeiras etapas. Isso ocorreu devido ao fato de que apenas com a leitura dos campos título, resumo e palavras-chaves notou-se que o trabalho fugiu dos objetivos deste mapeamento sistemático. Por exemplo, alguns trabalhos tinham como tema a virtualização, o processamento 3D, dispositivos móveis, entre outros assuntos que não se enquadram no que este trabalho busca. Ao fim dessa etapa tivemos uma redução de 184 para 22 trabalhos.

Na terceira e última etapa, foi realizada a leitura completa do artigo a fim de obter as respostas para as questões de pesquisas de cada trabalho selecionado na última etapa. Com a leitura integral dos trabalhos, foi possível finalizar o processo de filtragem, tendo em vista que os trabalhos foram integralmente confirmados através da leitura, que realmente o tema era condizente com o mapeamento sistemático e que ele possui informações suficientes para responder as questões de pesquisas. Nota-se que houve uma pequena redução entre estas etapas, de 22 para 19 trabalhos. somente 3 trabalhos foram removidos pois, ao realizar a leitura integral, notou-se que os mesmos não se encaixavam nos critérios de inclusão.

A distribuição dos 19 trabalhos selecionados em jornais, conferências e simpósios pode ser verificada na Figura 1.

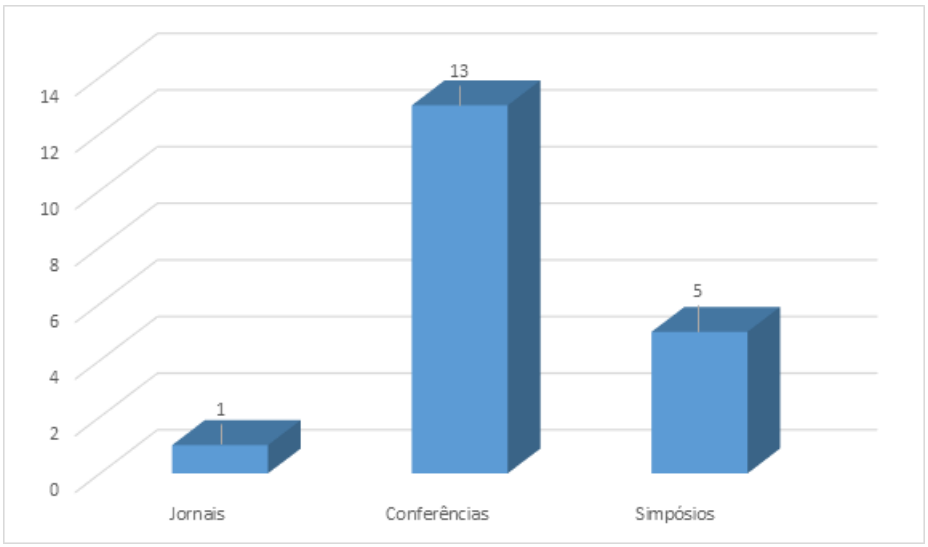

Figura 1. QP_1 - Distribuição dos trabalhos por jornais, conferências e simpósios.

A lista contendo os 19 trabalhos finais encontra-se AQUI..

4.1. Resultado das Questões de Pesquisa

Esta Seção apresenta os resultados obtidos com os estudos realizados considerando os 19 trabalhos resultantes da terceira etapa do mapeamento sistemático, fornecendo uma discussão sobre as 4 questões de pesquisas propostas neste trabalho.

QP_1 - A ferramenta realiza os *checkpoints* de maneira local ou global?

Como ilustrado na Figura 2, nota-se que existe uma grande diferença entre a quantidade de ferramentas que fazem o *checkpoint* de maneira global em relação a aos que fazem de maneira local. Dos trabalhos estudados, 16 deles apresentam ferramentas que executam *checkpoints* globais, enquanto que apenas 3 ([Fu et al. 2008, Subasi et al. 2015a, Ni et al. 2013]) executam *checkpoints* locais.

Segundo [Gabriel et al. 2009] e [Egwutuoha et al. 2013], um *checkpoint* local é uma "fotografia" do estado local de um processo. Mais precisamente, é um estado local de um processo que é escolhido para ser salvo em um meio de armazenamento estável para que, em caso de falhas de um processo, este seja restaurado até este estado salvo. Um *checkpoint* global consistente de uma aplicação de computação de alto desempenho é uma coleção de *checkpoints* locais, um por processo, de forma que esta coleção seja consistente. Um *checkpoint* global é consistente se seus *checkpoints* não possuem uma relação de precedência causal de [Lamport 1978].

Segundo [Chandy and Lamport 1985], um sistema distribuído consiste em um conjunto finito de processos e um conjunto finito de canais, ou seja os *checkpoints* globais naturalmente são realizados em ambientes com vários processos sendo executados simultaneamente. Isso reflete diretamente no número de trabalhos que utilizam o *checkpoint* global, pois devido a natureza dos sistemas distribuídos é mais simples a implementação de um coordenador global do que deixar que o processamento seja feito todo localmente.

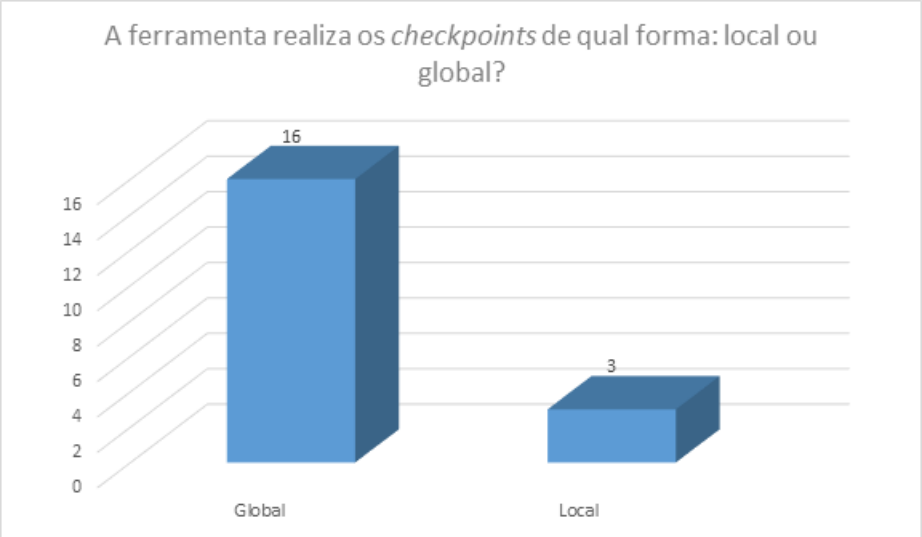

Figura 2. QP_1 - A ferramenta realiza os *checkpoints* de maneira local ou global?

QP_2 - A ferramenta é executada em qual nível: *user-level*, *kernel-level* ou *application-level*?

Para a discussão do resultado dessa questão de pesquisa é necessário compreender sobre o que se trata os três diferentes níveis de execução da ferramenta. Deste modo, segundo [Maeda 2002], programas de usuário estão no modo de usuário, que é o nível menos privilegiado, enquanto o nível *kernel* está no modo *kernel*, que é o mais nível privilegiado. Então, o *kernel* é mapeado na memória como uma região privilegiada que somente programas no modo *kernel* acessam. Os programas no modo do usuário não podem acessar o *kernel*. Neste sentido, o termo *kernel-level* significa que os *checkpoints*

executados são feitos em nível de *kernel*, ou seja, todas as informações de um processo, tais como a pilha de execução do processo, registradores, seu *heap*, entre outros recursos. O *user-level* consiste em retirar *checkpoints* no modo do usuário, onde este deve executar algum tipo de comando para que o *checkpoint*, local ou global, seja executado. *Checkpoints* em nível de aplicação (*application-level*) fica entre os outros dois em nível de privilégio e ferramentas que utilizam este tipo de *checkpoint* consideram que o *checkpoint* será feito dentro do código de uma aplicação de computação de alto desempenho.

O *kernel-level* apesar de possuir um nível de privilégio elevado, trás com ele alguns problema que acabam por fazer dele o menos escolhido pelos trabalhos, como o *kernel* está diretamente relacionado a arquitetura do sistema que está sendo executado, isso faz com que a portabilidade da ferramenta que é executada em *kernel-level* seja muito restrita, limitando a pouquíssimas possibilidades de execução em uma arquitetura diferente. Além de existir o problema de segurança, como dito por [Maeda 2002], é perigoso simplesmente permitir que os programas do usuário executem no modo *kernel* porque os programas podem operar todo o sistema sem restrições. Esse problema é ainda mais grave quando se trata de sistema distribuídos, pois é possível que seja necessário executar códigos em *kernels* diferentes.

Quando se trata de *user-level* o principal aspecto é a transparência e a portabilidade, pois devido ao seu nível de execução é possível fazer a portabilidade de maneira muito simples até mesmo para arquiteturas completamente diferentes e por ser executado no mesmo nível que o usuário opera, o usuário facilmente consegue identificar o que e como está sendo executado tornando o nível o mais transparente possível ao usuário. Por fim, em relação a segurança, ele não opera diretamente com o *kernel* limitando automaticamente seus acessos, porém isso gera um custo de processamento pois dependendo da necessidade é necessário realizar a comunição com o *kernel*.

Portanto, observa-se através da Figura 3 que o modo de execução de *checkpoints* é o *application-level*, pois ele é o balanceamento entre segurança, portabilidade e transparência. Esse conjunto o torna muito atrativo tanto para o desenvolvimento de ferramentas, quanto para a implementação das mesmas. Das 19 ferramentas estudadas, 11 delas consideram retirar *checkpoints* em nível de aplicação, enquanto que 7 delas retiram *checkpoints* em nível de usuário e apenas uma delas, [Rajachandrasekar et al. 2014], retira *checkpoints* em nível de *kernel*.

QP_3 - O *checkpoint* distribuído é feito manualmente ou automaticamente?

A principal diferença entre o processo manual para o automático é que, no processo automático a ferramenta faz o processo de *checkpoint*. Ferramentas que instrumentam *checkpoints* manualmente exigem que o usuário insira rotinas para que a aplicação execute *checkpoints*. Isso impõe a restrição de que o usuário deva ter um bom conhecimento do código da aplicação ou que este faça uma análise manual para determinar pontos onde *checkpoints* devam ser inseridos na aplicação. Todavia, ferramentas que instrumentam *checkpoints* automaticamente não exigem do usuário um conhecimento da aplicação, bastando apenas entender como usar a ferramenta de instrumentação. Este tipo de ferramenta, quando utilizada, faz o uso de mecanismos para determinar pontos da aplicação onde inserir rotinas de *checkpoint* e insere código automaticamente para o usuário [Rodríguez et al. 2010].

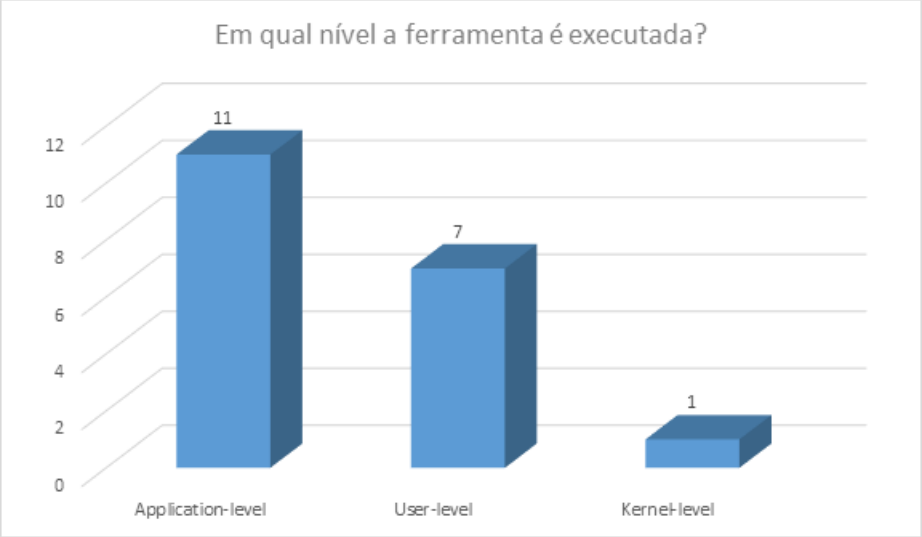

Figura 3. QP_2 **- A ferramenta é executada em qual nível: *user-level*, *kernel-level* ou *application-level*?**

De todas as 19 ferramentas estudadas, nota-se que 100% delas fazem o uso da instrumentação de *checkpoints* automaticamente, sugerindo que a utilização desse método não é apenas uma tendência ou preferência dos autores, e sim uma necessidade tendo em vista que, ao realizar o *checkpoint* manualmente, isso gera no usuário uma responsabilidade muito grande para que a ferramenta opere corretamente, o que pode ocasionar erro humano. Assim, para evitar esses problemas, todas as ferramentas selecionadas transferem a responsabilidade de realizar o *checkpoint* no tempo e de forma correta para a ferramenta em sí.

QP_4 **- O *checkpointing* é síncrono, assíncrono ou quase-síncrono?**

Protocolos de recuperação por retrocesso que usam o mecanismo de *ckeckpointing* pertencem a uma dentre três classes [Rodríguez et al. 2010]: síncronos, assíncronos ou quase-síncronos (ou induzidos por comunicação).

Protocolos assíncronos permitem que *checkpoints* sejam retirados de forma totalmente independente, ou seja, cada processo decide em que momento da execução de uma aplicação ele salva seu estado local. Por conta desta independência nesta classe de protocolos, os processos estão sujeitos ao efeito dominó [Randell 1975] e não garantem que a computação progrida.

Protocolos síncronos são os mais comuns na prática, principalmente devido à simplicidade na fase de recuperação. Nesta classe de protocolos de *checkpointing*, os processos precisam de um coordenador para gerenciar o momento em que um estado global seja determinado. Porém, de acordo com [Snir et al. 2014, Rodríguez et al. 2010], não escalam bem.

Protocolos induzidos por comunicação consideram o padrão de troca de mensagens entre os processos para salvar um estado de um processo. Nesses protocolos, *checkpoints* básicos são retirados de forma independente, como nos protocolos assíncronos. Contudo, *checkpoints* forçados são tomados de acordo com dados adicionais vindos em

mensagens da aplicação de forma a “quebrar” esta assincronia, permitindo a formação de estados globais consistentes e evitando o efeito dominó além de possuírem 2 desvantagens consideráveis que segundo [Egwutuoha et al. 2013]. são, é gerado um grande número de forçados que resultam em um no armazenamento e os dados que são adicionados nas mensagens geram um considerável na comunicação.

Ao analisar os resultados apresentados na Figura 4, dos 19 trabalhos estudados, 17 implementam protocolos síncronos, enquanto apenas 1 implementa protocolo assíncrono e 1 implementa protocolo quase-síncrono. Essa grande diferença entre o número de implementações síncronas para as demais comprova que realmente são as mais comuns devido à sua simplicidade de implementação e de recuperação, fazendo com que os autores optem por essa abordagem para que não haja dificuldades na implementação, evitando o efeito dominó causado por implementações assíncronas e também problemas de sincronismo em implementações quase-síncronas. A única ferramenta a utilizar protocolo assíncrono foi aquela descrita por [Subasi et al. 2015a], enquanto a única ferramenta a utilizar protocolo quase-sícrono foi a apresentada no trabalho de [Ghit and Epema 2017].

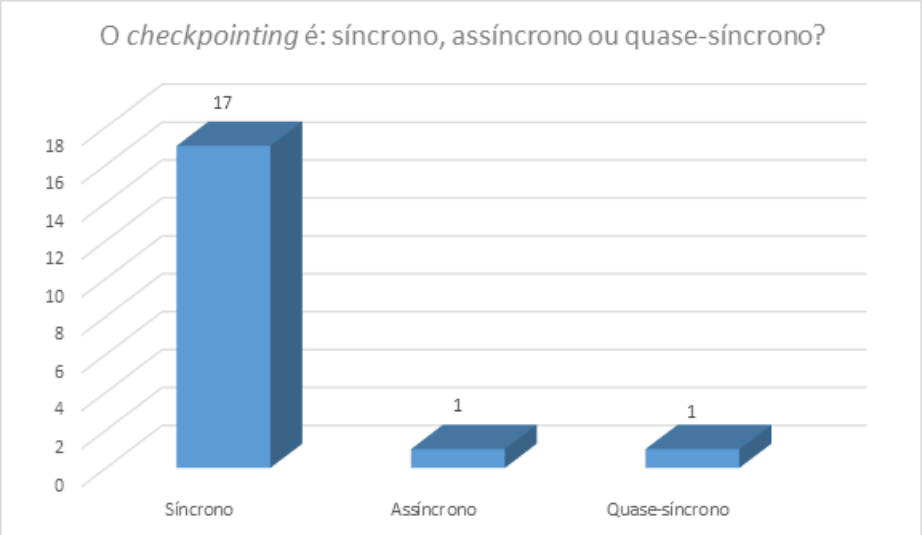

Figura 4. QP_4 - O *checkpointing* é síncrono, assíncrono ou quase-síncrono?

4.2. Discussão dos Resultados

Dentro dos resultados obtidos anteriormente das questões de pesquisas, obteve-se que em cada questão de pesquisa houve uma resposta que foi predominante, na QP_1 com 16 trabalhos foi a resposta ”Global”, na QP_2 com 11 trabalhos foi a resposta ”Application-level”, na QP_3 com 19 trabalhos foi a resposta ”Automaticamente”e por fim na QP_4 com 17 trabalhos foi a resposta ”Síncrono”. Respostas essas que são características das ferramentas, podendo os autores das ferramentas logicamente optarem por essas ou outras características no momento da implantação. Com esses dados pode-se analisar o seguinte, quantas ferramentas possuem simultaneamente as características citadas anteriormente, e o resultado foi de 9 ferramentas. Seguindo o mesmo processo, porém agora com um menor rigor, quantos trabalhos possuem simultaneamente 3 ou mais características das citadas anteriormente, e o resultado obtido foi de 17 ferramentas. Isso demonstra que apesar dessas características serem dominantes nas questões de pesquisas, uma implementação

que utilize simultaneamente todas elas não é algo comum, pois as ferramentas são criadas para um propósito que muitas das vezes necessita de uma ou mais característica(s) em específico.

Outro ponto importante, para um trabalho ser inserido dentro do mapeamento sistemático, considerou-se que o seu ano de publicação estaria entre os anos de 2000 e 2019. Após todas as etapas do mapeamento sistemático, fez-se um levantamento, apresentado na Figura 5, e obteve-se a quantidade de trabalhos por ano de publicação. Examinando esses resultados, percebe-se que o número de trabalhos publicados sobre ferramentas de *checkpointing* teve uma grande manifestação no ano de 2014. Além disso, mesmo considerando o intervalo de anos entre 2000 e 2019, somente entre os anos de 2004 e 2017 é que houveram trabalhos publicados sobre a construção dessas ferramentas.

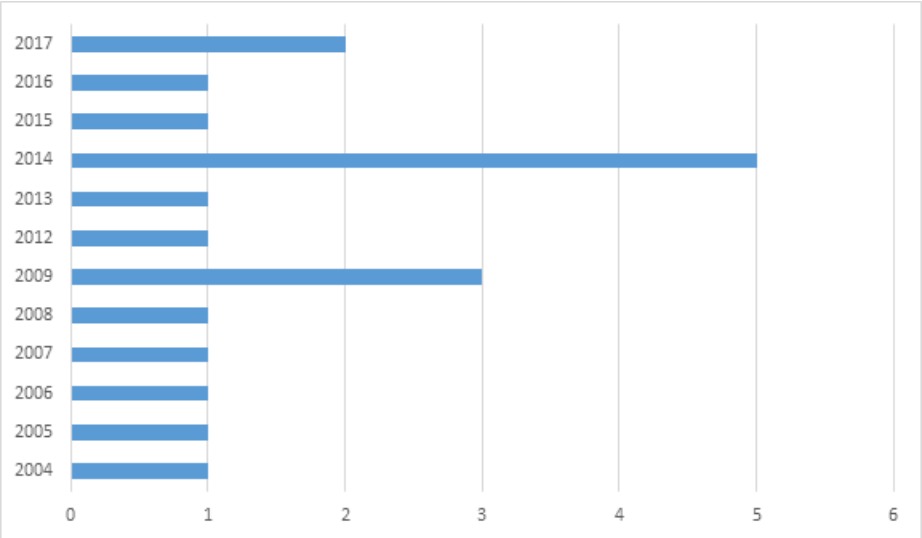

Figura 5. Quantidade de estudos por ano de publicação.

Conforme os dados obtidos no trabalho até o momento, pode-se fazer uma outra análise. Ao associar o ano de publicação dos trabalhos, conforme os dados apresentados na Figura 5 e as características que foram predominantes em suas respectivas questões de pesquisa, pode-se analisar se existe correlação entre as duas informações. Para isso foi realizado o seguinte processo, dividiu-se os anos de publicação dos trabalhos em 4 grupos contendo 5 trabalhos cada, com exceção do último grupo que ficou com 4 trabalhos. Esses grupos foram divididos da seguinte forma: trabalhos publicados entre os anos de 2004 e 2008, trabalhos entre 2009 e 2013, trabalhos entre 2014 e 2015 e entre 2016 e 2019. Após este processo, fez-se um levantamento de quantos trabalhos em cada grupo tinha como resposta as questões de pesquisa a alternativa que teve maior incidência. Aqui, a questão de pesquisa QP_3 foi descartada por possuir unanimidade em uma de suas alternativas.

Ao relacionar o ano de publicação dos trabalhos com o número de trabalho que possui como característica alguma das características predominantes nas questões de pesquisa (global, assíncrono e *application-level*) obteve-se o gráfico em bolhas apresentado na Figura 6 e, ao analisá-la, nota-se que não existe uma relação direta entre os anos e o aumento ou diminuição do número de trabalhos, pois nota-se que na QP_1 houve um aumento entre os anos de 2004 e 2015, após isso houve uma queda. Já ao analisar a QP_2

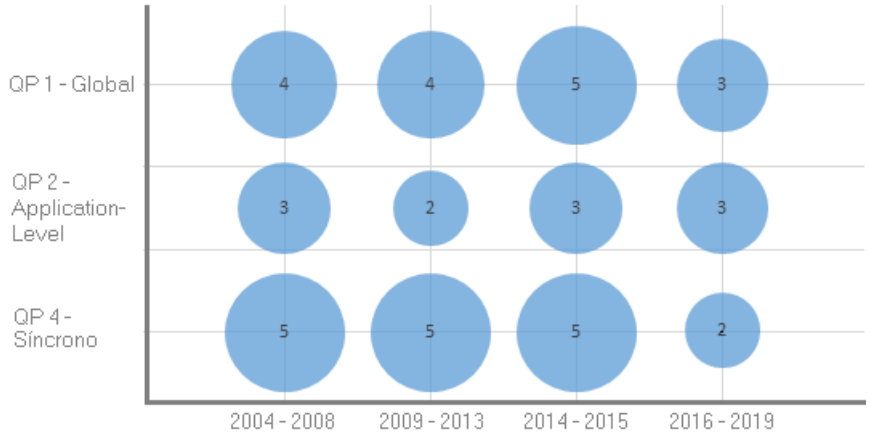

Figura 6. Mapeamento das respostas das questões de pesquisas de maior incidência (eixo y) e ano de publicação dos estudos (eixo x).

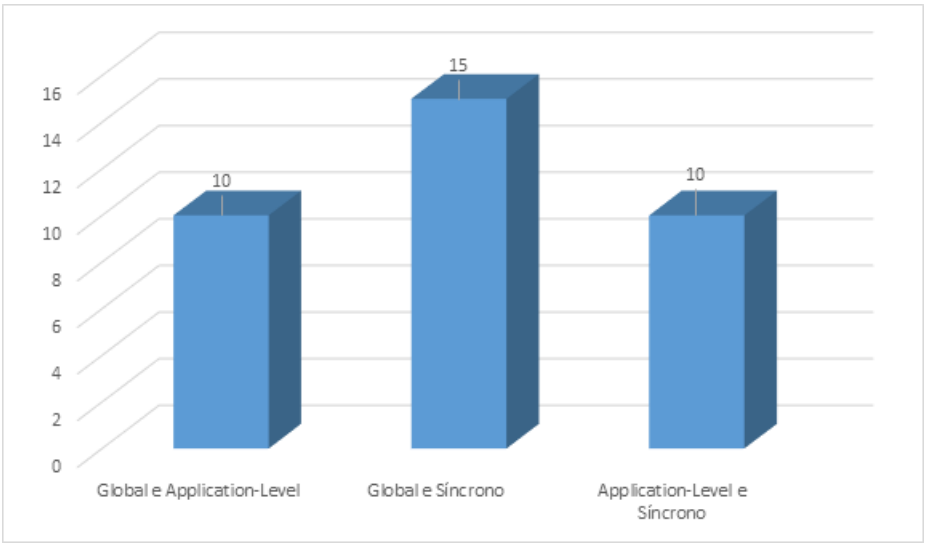

Figura 7. Combinação das questões de pesquisa.

nota-se que logo entre os anos de 2004 e 2009 houve uma redução. após isso houve um aumento entre 2014 e 2015, e posteriormente uma nova redução em 2016.

Por fim, pode-se realizar a análise da combinação das questões de pesquisas para levantar quantos trabalhos utilizam em conjunto as características mais utilizadas nos trabalhos, através da Figura 7 pode-se verificar que a quantidade de trabalhos que utilizam em conjuntos as características é alto, isso demostra que na literatura existem diversos trabalhos que abordam a implementação utilizando essas características.

5. Considerações Finais

Dos 19 trabalhos analisados, constatou-se que a maioria dos trabalhos são implementados de forma conservadora tendo em vista que 17 trabalhos utilizam pelo menos 3 das características mais utilizadas, pois em sua implementação são utilizados métodos mais simples e que possuam na literatura estudos que incentivem essas implementações, como é o caso do sincronismo. Dessa forma são poucas as ferramentas que partem para

uma implementação pouco convencional, tornando as dificuldades de se implantar algo pouco comum ainda maiores pelo fato de existirem poucos trabalhos relacionados. Este mapeamento sistemática demonstrou que existem diversas ferramentas que auxiliam na instrumentação de *checkpoints* nas mais variadas implementações, tendo como principal ponto as repostas as questões de pesquisas, as quais demonstram de maneira precisa as tendência tomadas pelos autores das ferramentas, demonstrando que existe uma deficiência em relação ao número de trabalhos que utilizam uma implementação de protocolo assíncrono e quase-síncrono assim como implementações com *checkpoint* manual. Por fim, através deste mapeamento sistemático é possível notar que o maior número de trabalhos foi no ano de 2014, justamente onde a maioria dos trabalhos usaram as implementações que tiveram maior incidência neste trabalho.

Durante o desenvolvimento do mapeamento sistemática foram encontradas algumas dificuldades, a primeira dificuldade ocorreu no momento da elaboração das *strings* de busca, pelo fato de que uma pequena mudança poderia mudar drasticamente o resultado das buscas, podendo fazer com que o resultado ficasse fora do esperado. A segunda dificuldade foi encontrada no momento busca pelas respostas as perguntas de pesquisas, levando em consideração que nem todos os 19 trabalhos da etapa final respondiam com clareza as questões de perguntas, fez-se necessário avaliar criteriosamente as informações fornecidas no trabalho para se obter a resposta. Como última dificuldade, encontrar trabalhos recentes de mapeamento sistemático com tema ou área relacionadas a este trabalho.

Como dito anteriormente, são poucos os estudos de mapeamento sistemático aplicados a área de computação distribuída, por este motivo ainda existe uma área muito grande a ser pesquisada, no futuro com o aumento cada vez maior dos sistemas distribuídos os trabalhos futuros poderão serem focados em como as implementações síncronas irão se comportar em sistemas muito grandes, tendo em vista que um de seus pontos fracos é justamente não escalarem bem. Ou até mesmo uma possível ferramenta com implementação assíncrona que possua uma solução para o efeito dominó e que por consequência escale melhor que as implementações síncronas em sistemas muito grandes.

Referências

[Abreu et al. 2012] Abreu, F., Almeida, A., Barreiros, E., Saraiva, J., Soares, S., Araújo, A., and Henrique, G. (2012). Métodos, técnicas e ferramentas para o desenvolvimento de software educacional: Um mapeamento sistemático. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, 23(1).

[Almeida et al. 2011] Almeida, A., Barreiros, E., Saraiva, J., and Soares, S. (2011). Mecanismos para guiar estudos empíricos em engenharia de software: Um mapeamento sistemático. In *Proceedings of the Experimental Software Engineering Latin American Workshop (ESELAW)*. Rio de Janeiro, Brazil.

[Ansel et al. 2009] Ansel, J., Arya, K., and Cooperman, G. (2009). Dmtcp: Transparent checkpointing for cluster computations and the desktop. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12.

[Armanda et al. 2012] Armanda, M. d. A., Rodrigues, R. L., and Garcia, V. C. (2012). Um mapeamento sistemático para problem based learning aplicado à ciência da computação. *Anais do Workshop de Informática na Escola*, 1(1).

[Baldoni et al. 1997] Baldoni, R., Helary, J.-M., Mostefaoui, A., and Raynal, M. (1997). A communication-induced checkpointing protocol that ensures rollback-dependency trackability. In *Fault-Tolerant Computing, 1997. FTCS-27. Digest of Papers., Twenty-Seventh Annual International Symposium on*, pages 68–77.

[Borges et al. 2013] Borges, S. d. S., Reis, H. M., Durelli, V. H., Bittencourt, I. I., Jaques, P. A., and Isotani, S. (2013). Gamificação aplicada à educação: um mapeamento sistemático. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, 24(1):234.

[Cao and Singhal 2001] Cao, G. and Singhal, M. (2001). Mutable checkpoints: a new checkpointing approach for mobile computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 12(2):157–172.

[Cao and Singhal 2003] Cao, G. and Singhal, M. (2003). Checkpointing with mutable checkpoints. *Theoretical Computer Science*, 290(2):1127–1148. Dependable Computing.

[Cao et al. 2014] Cao, J., Kerr, G., Arya, K., and Cooperman, G. (2014). Transparent checkpoint-restart over infiniband. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 13–24, New York, NY, USA. ACM.

[Cappello et al. 2014] Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., and Snir, M. (2014). Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1).

[Chandy and Lamport 1985] Chandy, K. M. and Lamport, L. (1985). Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75.

[Egwutuoha et al. 2013] Egwutuoha, I., Levy, D., Selic, B., and Chen, S. (2013). A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65(3):1302–1326.

[Elnozahy et al. 2002] Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34:375–408.

[Fu et al. 2008] Fu, H., Du, Y., Wang, P., Jia, J., and Yang, X. (2008). Gift: Automating ftpa implementation for mpi programs. In *2008 14th IEEE International Conference on Parallel and Distributed Systems*, pages 91–98.

[Gabriel et al. 2009] Gabriel, R., J., M. M., Patricia, G., Juan, T., and Ramón, D. (2009). Cppc: a compiler-assisted tool for portable checkpointing of message-passing applications. *Concurrency and Computation: Practice and Experience*, 22(6):749–766.

[Gamell et al. 2014] Gamell, M., Katz, D., Kolla, H., Chen, J., Klasky, S., and Parashar, M. (2014). Exploring automatic, online failure recovery for scientific applications at extreme scales. *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[Gao et al. 2006] Gao, Q., Yu, W., Huang, W., and Panda, D. K. (2006). Application-transparent checkpoint/restart for mpi programs over infiniband. In *2006 International Conference on Parallel Processing (ICPP'06)*, pages 471–478.

[Garcia and Buzzato 2001] Garcia, I. and Buzzato, L. (2001). On the minimal characterization of the rollback-dependency trackability property. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 342–349.

[Garcia et al. 2001] Garcia, I., Vieira, G., and Buzato, L. (2001). Rdt-partner: An efficient checkpointing protocol that enforces rollback-dependency trackability. In *Proc. 19th Brazilian Symposium Computer Networks*.

[Garg et al. 2010] Garg, R., Garg, V., and Sabharwal, Y. (2010). Efficient algorithms for global snapshots in large distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):620 –630.

[Gärtner 1999] Gärtner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26.

[Ghit and Epema 2017] Ghit, B. and Epema, D. (2017). Better safe than sorry: Grappling with failures of in-memory data analytics frameworks. In *HPDC '17 Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pages 105–116.

[Hursey et al. 2009] Hursey, J., Mattox, T. I., and Lumsdaine, A. (2009). Interconnect agnostic checkpoint/restart in open mpi. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, pages 49–58, New York, NY, USA. ACM.

[Hursey et al. 2007] Hursey, J., Squyres, J. M., Mattox, T. I., and Lumsdaine, A. (2007). The design and implementation of checkpoint/restart process fault tolerance for open mpi. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8.

[Hélary et al. 2000] Hélary, J.-M., Mostefaoui, A., Netzer, R., and Raynal, M. (2000). Communication-based prevention of useless checkpoints in distributed computations. *Distributed Computing*, 13:29–43. 10.1007/s004460050003.

[Kalaiselvi and Rajaraman 2000] Kalaiselvi, S. and Rajaraman, V. (2000). A survey of checkpointing algorithms for parallel and distributed computers. *Sadhana (Academy Proceedings in Engineering Sciences)*, 25(5):489–510.

[Kshemkalyani 2009] Kshemkalyani, A. (2009). A symmetric o(n log n) message distributed snapshot algorithm for large-scale systems. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1 –4.

[Kshemkalyani 2010] Kshemkalyani, A. (2010). Fast and message-efficient global snapshot algorithms for large-scale distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 21(9):1281 –1289.

[Lamport 1978] Lamport, L. (1978). Times, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21:558–565.

[Losada et al. 2017] Losada, N., Fraguela, B. B., González, P., and Martín, M. J. (2017). A portable and adaptable fault tolerance solution for heterogeneous applications. *Journal of Parallel and Distributed Computing*, 104:146 – 158.

[Losada et al. 2016] Losada, N., Martín, M. J., Rodríguez, G., and González, P. (2016). Portable application-level checkpointing for hybrid mpi-openmp applications. *Procedia Computer Science*, 80:19 – 29. International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[Luo and Manivannan 2009] Luo, Y. and Manivannan, D. (2009). Fine: A fully informed and efficient communication-induced checkpointing protocol for distributed systems. *Journal of Parallel and Distributed Computing*, 69(2):153–167.

[Ma et al. 2009] Ma, C., Huo, Z., Cai, J., and Meng, D. (2009). Dcr: A fully transparent checkpoint/restart framework for distributed systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10.

[Maeda 2002] Maeda, T. (2002). Safe execution of user programs in kernel mode using typed assembly language. Master's thesis, The University of Tokyo.

[Magalhães et al. 2013] Magalhães, C. V., Santos, R. E., Da Silva, F. Q., and Gomes, A. S. (2013). Caracterizando a pesquisa em informática na educação no brasil: um mapeamento sistemático das publicações do sbie. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, 24(1):22.

[Makassikis et al. 2012] Makassikis, C., Vialle, S., and Warin, X. (2012). Ft-grelosss: A skeletal-based approach towards application parallelization and low-overhead fault tolerance. *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*.

[Manivannan and Singhal 1996] Manivannan, D. and Singhal, M. (1996). Quasi-synchronous checkpointing: Models, characterization and classification. Technical Report OSU-CISRC-5/96-TR33, Ohio State University, Computer Science and Engineering Research Center, Department of Computer Science and Engineering.

[Ni et al. 2013] Ni, X., Meneses, E., Jain, N., and Kalé, L. V. (2013). Acr: Automatic checkpoint/restart for soft and hard error protection. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 7:1–7:12, New York, NY, USA. ACM.

[Peixoto et al. 2015] Peixoto, M., Silva, C., Vilela, J., and Gonçalves, E. (2015). Um mapeamento sistemático de gamificação em software educativo no contexto da comunidade brasileira de informática na educação. *Anais do Workshop de Informática na Escola*, 21(1):584.

[Petersen et al. 2008] Petersen, K., Feldt, R., S., M., and Mattsson, M. (2008). Systematic mapping studies in software engineering. *Systematic mapping studies in software engineering*, 17:1.

[Petersen et al. 2015] Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

[Rahman 2012] Rahman, M. M. (2012). Process synchronization in multiprocessor and multi-core processor. *2012 International Conference on Informatics, Electronics & Vision (ICIEV)*.

[Rajachandrasekar et al. 2014] Rajachandrasekar, R., Potluri, S., Venkatesh, A., Hamidouche, K., Wasi-ur Rahman, M., and Panda, D. K. D. (2014). Mic-check: A distributed check pointing framework for the intel many integrated cores architecture. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 121–124, New York, NY, USA. ACM.

[Randell 1975] Randell, B. (1975). System structure for software fault tolerance. *Ieee transactions on software engineering*, (2):220–232.

[Rezaei et al. 2014] Rezaei, A., Coviello, G., Li, C.-H., Chakradhar, S., and Mueller, F. (2014). Snapify: Capturing snapshots of offload applications on xeon phi manycore processors. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 1–12, New York, NY, USA. ACM.

[Rodríguez et al. 2010] Rodríguez, G., Martín, M. J., González, P., Touriño, J., and Doallo, R. (2010). CPPC: a compiler-assisted tool for portable checkpointing of message-passing applications. *Concurrency and Computation: Practice and Experience*, 22(6):749–766.

[Ruscio et al. 2007] Ruscio, J., Heffner, M., and Varadarajan, S. (2007). Dejavu: Transparent user-level checkpointing, migration, and recovery for distributed systems. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10.

[Shahzad et al. 2013] Shahzad, F., Wittmann, M., Kreutzer, M., Zeiser, T., Hager, G., and Wellein, G. (2013). A survey of checkpoint/restart techniques on distributed memory systems. *Parallel Processing Letters*, 23(04):1340011.

[Snir et al. 2014] Snir, M., Wisniewski, R. W., Abraham, J. A., Adve, S. V., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., Chien, A. A., Coteus, P., DeBardeleben, N. A., Diniz, P. C., Engelmann, C., Erez, M., Fazzari, S., Geist, A., Gupta, R., Johnson, F., Krishnamoorthy, S., Leyffer, S., Liberty, D., Mitra, S., Munson, T., Schreiber, R., Stearley, J., and Hensbergen, E. V. (2014). Addressing failures in exascale computing. *International Journal of High Performance Computing Applications*, 28(2):129–173.

[Sorin et al. 2002] Sorin, D. J., Milo, M. K. M., Hill, M. D., and Wood, D. A. (2002). Safetynet: Improving the availability of shared memory multiprocessors with global checkpoint/recovery. *29th Annual International Symposium on Computer Architecture*.

[Subasi et al. 2015a] Subasi, O., Arias, J., Unsal, O., Labarta, J., and Cristal, A. (2015a). Nanocheckpoints: A task-based asynchronous dataflow framework for efficient and scalable checkpoint/restart. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 99–102.

[Subasi et al. 2015b] Subasi, O., Arias, J., Unsal, O., Labarta, J., and Cristal, A. (2015b). Nanocheckpoints: A task-based asynchronous dataflow framework for efficient and

scalable checkpoint/restart. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 99–102.

[Treaster 2005] Treaster, M. (2005). A survey of fault-tolerance and fault-recovery techniques in parallel systems. *eprint arXiv:cs/0501002*.

[Tsai and Lin 2005] Tsai, J. and Lin, J.-W. (2005). On the fully-informed communication-induced checkpointing protocol. In *Dependable Computing, 2005. Proceedings. 11th Pacific Rim International Symposium on*, page 8 pages.

[Vieira et al. 2001] Vieira, G., Garcia, I., and Buzato, L. (2001). Systematic analysis of index-based checkpointing algorithms using simulation. In *Proceedings of IX Brazilian Symposium on Fault-Tolerant Computing*, pages 31–41.

[Wang 1997] Wang, Y.-M. (1997). Consistent global checkpoints that contain a given set of local checkpoints. *Computers, IEEE Transactions on*, 46(4):456 –468.