Impacto do Custo de Salvamento e dos Atributos de Configuração no *Checkpoint* do Apache Hadoop

Paulo V. M. Cardoso[1], **Patrícia Pitthan Barcelos**[1]

[1]Pós-Graduação em Ciência da Computação (PGCC)
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

pcardoso@inf.ufsm.br, pitthan@inf.ufsm.br

Abstract. *The Apache Hadoop framework, created to process and to store large amounts of data, implements the Checkpoint and Recovery technique for efficient failure recoveries in its distributed file system. However, proper adaptions of period between Hadoop checkpoints depend on accurated observations of system elements. This work relies on estimating checkpoint procedure cost and mean time between failures cost based on historical records. The factors are observed and analysed concerning to different attribute configurations of both Hadoop and tested benchmark.*

Resumo. *O framework Apache Hadoop, usado para processar e armazenar grandes quantidades de dados, usa a técnica de Checkpoint and Recovery para auxiliar em recuperações pós-falha de seu sistema de arquivos distribuído. Porém, adaptações eficientes para período entre checkpoint do Hadoop dependem de observações apuradas do sistema. O objetivo deste trabalho é estimar o custo da realização de checkpoints e do tempo médio entre falhas do sistema a partir de um histórico de observações. Os fatores são observados e analisados com relação a diferentes variações de configuração do framework e do benchmark usado.*

1. Introdução

A necessidade de armazenamento e processamento de quantidades massivas de dados é uma característica cada vez mais presente no contexto computacional. Para atender a essa demanda, o uso de sistemas computacionais que ofereçam alto desempenho e alta confiabilidade são essenciais. Como a presença de falhas nesses sistemas é inevitável [Ghit and Epema 2017], seus mecanismos de tolerância a falhas devem providenciar recuperações rápidas ao mesmo tempo em que devem evitar a intrusividade.

Uma técnica amplamente usada no contexto de tolerância a falhas em sistemas distribuídos é a recuperação de erros por retorno, em que o andamento de um sistema é retrocedido para um estado estável anterior [Laprie 1985]. Nesse cenário, uma das principais alternativas de recuperação por retorno é o *Checkpoint and Recovery* (CR, ou apenas *checkpoint*), que consiste em salvar o estado do sistema em arquivos de *checkpoint* de forma preventiva, enquanto não houver ocorrência de falhas. Após a detecção de uma falha, o CR realiza a recuperação a partir da reconstrução do sistema baseado no último *checkpoint* salvo.

Variações da técnica de CR são encontradas em diversas ferramentas voltadas para processamento de alto desempenho. Uma dessas ferramentas é o Apache Hadoop: um *framework* de alto desempenho desenvolvido para o processamento e o armazenamento de *Big Data*. No Hadoop, o *checkpoint* é realizado periodicamente para salvar o estado atual do seu sistema de arquivos distribuído (*Hadoop Distributed File System* – HDFS).

Visto que o Hadoop tem capacidade para lidar com diferentes tipos de aplicações (muitas vezes de forma simultânea no mesmo ambiente de execução), o sistema pode sofrer com alterações constantes de comportamento. Nesse sentido, a configuração de *checkpoints* usada pelo *framework* gera uma limitação ao não permitir alterações em tempo real. Isto é, o período entre *checkpoints* é definido antes do início da execução do Hadoop e só pode ser alterado com a interrupção do próprio *framework* e de todos os seus serviços. Assim, as alterações de comportamento não refletidas em modificações no período podem afetar a intrusividade e a confiabilidade.

Uma alternativa que já se mostrou eficiente nesse contexto é o mecanismo de configuração dinâmica para atributos de *checkpoint* [Cardoso and Barcelos 2018b] [Cardoso and Barcelos 2018a]. O mecanismo dinâmico permite que alterações em tempo real do período entre *checkpoints* seja possível no âmbito do Hadoop. Essas mudanças são automáticas e definidas por métricas de monitoramento baseadas no comportamento do sistema. Assim, a partir da análise de utilização do sistema, aproximações de períodos ideais podem ser calculadas para otimizar o desempenho da técnica de *checkpoint*.

Porém, a análise de elementos do sistema pode se tornar uma tarefa complexa à medida que informações sobre o ambiente computacional são desconhecidas ou requerem um conhecimento *a-priori* de fatores geralmente não disponíveis ao desenvolvedor. Desta forma, o objetivo deste trabalho é analisar dois fatores diretamente ligados à eficiência da técnica de *checkpoint*: o custo de salvamento dos *checkpoints* e o nível de confiabilidade do sistema, calculado pelo tempo médio entre falhas.

Para mensurar essas informações – inicialmente desconhecidas –, é usada a funcionalidade de histórico do mecanismo de configuração dinâmica, que adapta os valores em tempo real conforme novos dados são obtidos. Para criar um cenário de avaliação da técnica de *checkpoint*, foram submetidos testes no *benchmark* $TestDFSIO$ [Noll 2011] com a definição de cenários de falhas transientes induzidas no elemento mestre do HDFS. Além disso, para uma análise de custos mais apurada, também foram criados diferentes cenários de configuração do período entre *checkpoints* (usando-se as abordagens estática e dinâmica) e de carga dos dados no *benchmark* utilizado, a fim de evidenciar a relação entre essas variações e os custos observados.

O artigo está estruturado da seguinte forma: a Seção 2 apresenta o *framework* Apache Hadoop e seu sistema de arquivos distribuído. Na seção 3, é apresentado o uso da técnica de *checkpoint* no HDFS através dos mecanismos de configuração estática e dinâmica. A Seção 4 descreve a arquitetura de histórico para análises de custos. Na Seção 5, apontam-se os experimentos e a discussão acerca dos resultados. A Seção 6 apresenta conclusões e próximos passos.

2. Apache Hadoop

O Apache Hadoop é um projeto *open source* voltado para o processamento distribuído de grandes quantidades de dados [White 2015]. O Hadoop oferece um eficiente sistema

de distribuição de dados e aplicações em arquiteturas de alto desempenho, como *clusters* e *grids*. O conceito essencial do Hadoop é mover a aplicação até os dados – evitando mover os dados em si. A arquitetura do Hadoop, em sua versão v2.7.3, é composta por diversos módulos, dentre os quais destacam-se: o sistema de arquivos distribuído (HDFS) e o *framework* para manutenção do ambiente e da aplicação (YARN).

Os dados no Apache Hadoop são armazenados em um sistema de arquivos distribuído denominado *Hadoop Distributed File System* (HDFS), que foi projetado para oferecer suporte a arquivos de grandes dimensões. Um arquivo no HDFS é dividido em blocos de tamanho pré-definido e distribuídos através do ambiente. A distribuição faz com que blocos de um mesmo arquivo possam estar armazenados em diferentes nós, para que o propósito de mover a computação até os nós, implementado pelo Hadoop, seja facilitado.

No HDFS, dados e metadados são armazenados de forma separada. Os metadados são mantidos por um servidor dedicado, chamado $NameNode$ (NN), enquanto um $DataNode$ (DN) é responsável por realizar o armazenamento dos dados. A arquitetura do HDFS é baseada no modelo *1-master N-workers*, em que um único NN serve N $DataNodes$ [White 2015]. O $NameNode$ também atende às requisições de aplicações e gerencia os arquivos e $DataNodes$ do HDFS.

A representação de arquivos e diretórios no $NameNode$ é realizada por objetos do tipo `INode`, que são armazenados em sua memória local, tornando disponível um mapeamento de arquivos/blocos no HDFS. Para completar o *namespace*, há um outro mapeamento que possui informações sobre a localização dos blocos de cada DN. Em um intervalo de tempo pré-definido (por padrão 4 segundos), os $DataNodes$ realizam uma varredura em seus blocos e reportam o estado para o $NameNode$. Assim, o nó mestre do HDFS possui uma visão sobre a localização de cada bloco do HDFS.

3. Checkpoint

Para garantir confiabilidade e disponibilidade, o Hadoop possui implementações das seguintes técnicas de tolerância a falhas: replicação de blocos, mensagens *heartbeat* e *checkpoint*. Enquanto a replicação consiste em armazenar réplicas de blocos no HDFS para assegurar disponibilidade de dados, as mensagens *heartbeat* são usadas como meio de comunicação do NN com os DNs, além de servir como forma de detecção de falhas. Por outro lado, a técnica de *checkpoint* visa garantir a confiabilidade de execução do NN.

Checkpoint and Recovery (CR) consiste em uma técnica de tolerância a falhas reativa, classificada como técnica de recuperação por retorno (*backward error recovery*). O CR tem como ideia principal a recuperação do estado falho de um sistema através de um contexto estável previamente salvo [Egwutuoha et al. 2013]. Essa recuperação busca prevenir erros computacionais consequentes das falhas. Os contextos estáveis são salvos periodicamente e armazenados de forma segura para posterior recuperação do sistema.

Os casos em que a técnica de *checkpoint* se mostra vantajosa estão relacionados a aplicações de longa duração (*long-running applications*) [Cui et al. 2015], frequentemente executadas pelo Hadoop. O tempo de execução dessas aplicações pode girar em torno de horas ou até dias. Outro cenário favorável ao uso de CR é o processamento intensivo de dados, devido ao grande número de operações de *I/O*. Nesses casos, uma falha ao final da execução pode acarretar em uma perda total ou reexecução completa.

O *checkpoint* no Hadoop é implementado para tolerar falhas no $NameNode$. Apesar de não ser o principal mecanismo de tolerância a falhas do Hadoop, o salvamento de *checkpoints* é essencial para a manutenção do $NameNode$ e de seus metadados. Assim, o *namespace* do HDFS é replicado em um arquivo chamado `FSImage`, armazenado em disco no sistema de arquivos local do NN. Nesse arquivo, são mantidas informações sobre o mapeamento de blocos e propriedades do sistema.

Para evitar a criação de um novo `FSImage` a cada operação realizada, um *log* de edições (`EditLog`), também mantido em disco local, armazena as últimas transações efetuadas após a criação do `FSImage`. O *merge* entre o `FSImage` e o `EditLog` consiste no estabelecimento do *checkpoint*. Esse procedimento é feito de forma periódica ou quando o HDFS atinge um número específico de transações [White 2015]. Como exceção, o arquivo de checkpoint também é salvo na inicialização do NN. Nesse momento, o salvamento de um checkpoint é usado para reestabelecer a condição e o andamento estável do NN após um reinício.

Para que o $NameNode$ não seja interrompido durante um *checkpoint*, o Hadoop executa um elemento chamado $SecondaryNameNode$ (SNN). A função do SNN é realizar o procedimento de *checkpoint* assim que o NN requisita. A cada salvamento de *checkpoint*, o NN transfere o `EditLog` para o SNN, que mantém uma cópia do `FSImage`. Assim, o *merge* entre os arquivos é feito para que o novo `FSImage` seja criado. Uma cópia do novo arquivo é enviada de volta ao NN e outra é armazenada no SNN.

3.1. Configuração estática

Nas versões *default* do Hadoop, o intervalo de tempo entre dois *checkpoints* consecutivos é definido de forma estática no arquivo de configuração $hdfs - site.xml$. O período a ser usado deve ser especificado antes do início do *framework*. Caso nenhum atributo seja definido nos arquivos de configuração, o valor padrão de 3600 segundos é usado.

A característica de configuração estática não suporta alterações nos atributos de *checkpoint* em tempo de execução. Ainda que uma modificação no arquivo de configuração seja permitida durante a execução do Hadoop, sua aplicação na prática somente é efetivada a partir do reinicio do *framework*. Desta forma, a modificação dos atributos de *checkpoint* implementada pelo *framework* é limitada, uma vez que se faz necessário interromper seu funcionamento para que qualquer tipo de alteração tenha efeito.

Visto que diferentes aplicações possuem demandas exclusivas, o procedimento de *checkpoint* pode se comportar de maneira distinta em cada situação. Por isso, a escolha do intervalo ideal entre *checkpoints* – que ofereça uma alta confiabilidade mas não interfira no desempenho das aplicações – consiste em um grande desafio. Intervalos estáticos, que são definidos de forma prévia e não mudam no decorrer da execução, tornam-se prejudiciais pois não oferecem suporte às mudanças de comportamento do ambiente.

3.2. Configuração Dinâmica

O *checkpoint* apresenta-se como um importante mecanismo no contexto da tolerância a falhas. Porém, configurá-lo de forma eficiente é um grande desafio, uma vez que diversos fatores podem interferir no seu comportamento. Sob escolhas inapropriadas, o sistema e as aplicações sofrem com problemas de confiabilidade e desempenho. Nesse caso, a propriedade de tolerância a falhas do *checkpoint* perde o seu propósito.

Desta forma, a arquitetura de configuração dinâmica para atributos de *checkpoint* [Cardoso and Barcelos 2018b] tem por objetivo tornar mais eficiente a configuração de atributos relacionados ao mecanismo. Com configurações dinâmicas, novos atributos são calculados em tempo real, com um foco voltado a propriedades adequadas para determinados momentos. Para isso, dois elementos essenciais são usados: o monitor e o coordenador, exibidos na Figura 1.

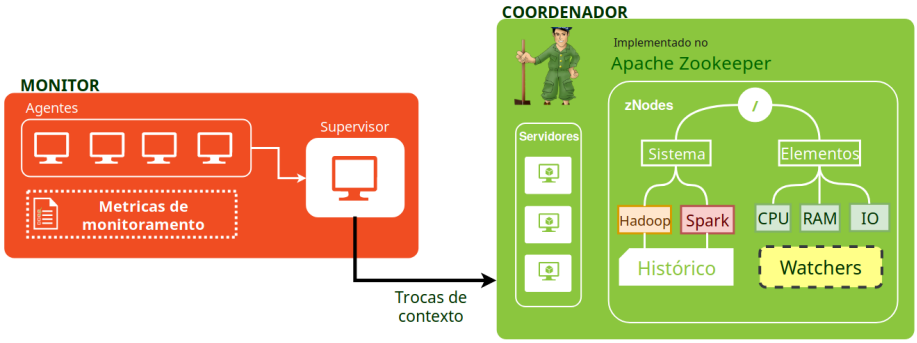


Figura 1. Arquitetura de configuração dinâmica para o *checkpoint* no HDFS.

3.2.1. Monitor

O monitoramento de recursos é uma etapa essencial da definição dinâmica de períodos entre *checkpoints*. A partir de análises sobre a situação do sistema, é possível realizar uma quantificação da utilização de recursos. Assim, a arquitetura dinâmica tem a possibilidade de identificar a necessidade de adaptação do período entre *checkpoints*. Assim como a maioria das ferramentas de monitoramento de recursos a arquitetura do módulo monitor deste trabalho utiliza uma implementação agente-servidor.

Os agentes – executados individualmente em nós do ambiente – coletam informações sobre recursos e notificam o servidor quando há uma alteração significativa de comportamento. Os fatores de utilização passíveis de observação são os seguintes: (a) CPU, (b) memória RAM e (c) operações em disco. Por sua vez, o servidor (supervisor) mantém uma comunicação passiva com os agentes. Quando uma mensagem é recebida, o supervisor armazena as informações e faz uma análise dos dados de agentes.

Um cálculo que indica novos atributos de *checkpoint* a serem usados é feito pelo supervisor e o novo valor é atualizado no coordenador. A relevância da alteração de comportamento nos monitores, bem como a política adotada para calcular um novo período entre *checkpoints*, são definidas de acordo com métricas de monitoramento. Nesse caso, aproximações já conhecidas para cálculo de períodos otimizados, como as fórmulas de Young [Young 1974] e Daly [Daly 2006], podem ser usadas pelo supervisor.

3.2.2. Coordenador

Visando um armazenamento seguro de informações, o coordenador foi usado como um repositório central de atributos de configuração, a partir do *framework* Apache Zookeeper: um projeto *open source* com funcionalidades para facilitar a coordenação de sistemas

distribuídos [Hunt et al. 2010]. A arquitetura do Zookeeper é formada por servidores que realizam operações requisitadas por clientes em um *namespace* compartilhado, cuja estrutura é composta por uma árvore de $zNodes$.

Nesse *framework* foram criados diversos atributos de configuração em $zNodes$. O atributo atual é definido em um $zNode$ específico, sendo que os atributos já usados são mantidos em um histórico. Uma importante função do coordenador é a configuração de mecanismos de alerta (*watchers*), que geram notificações para cada modificação em um $zNode$. O nó responsável por armazenar o atributo de configuração atual possui um *watcher* que direciona a uma mudança para qualquer elemento que o esteja observando.

3.2.3. Implementação no HDFS

O processo de *checkpoint* pode se tornar um fator crítico para o desempenho do Apache Hadoop. A escolha por períodos frequentes ou mais longos implica em diferentes comportamentos de recuperação e, inclusive, pode prejudicar o andamento de aplicações.

A implementação do mecanismo dinâmico no HDFS tem como objetivo a adaptação em tempo real do período entre *checkpoints* [Cardoso and Barcelos 2018b]. As modificações são feitas sem a necessidade de interromper o Hadoop e seus serviços. Para isso, foram alterados os seguintes elementos essenciais para o *checkpoint* no SNN: o `CheckpointConf` ($CConf$), que fornece um acesso em memória aos dados de configuração, e o `Checkpointer` ($Cptr$), que lidera o procedimento de *checkpoint*.

No mecanismo dinâmico foram realizadas alterações no tratamento do período entre *checkpoints* no $Cptr$. Essas alterações incluem um elemento de comunicação entre o HDFS e o módulo coordenador e um tratamento de alertas, para alterações nos atributos de configuração em uma troca de contexto. Caso o período seja modificado, o $Cptr$ deve verificar se o novo intervalo já foi atingido na espera da antiga configuração, invocando um novo salvamento de *checkpoint* caso este fato ocorra, ou esperando o tempo restante.

A modificação no $Cptr$ é realizada a partir da inicialização da classe e do procedimento de *checkpoint*. Ao iniciar, a classe verifica a abordagem de configuração do período entre *checkpoints* a ser utilizada (dinâmica ou estática), conectando-se ao coordenador se necessário. Logo, a classe $CConf$ é criada e o salvamento de *checkpoints* é iniciado. Caso uma mudança de atributo aconteça durante uma espera, o processo reinicia imediatamente para calcular o período restante. O *checkpoint* ocorre quando nenhuma espera é necessária.

4. Histórico de atributos

Para proporcionar uma visão completa do comportamento dos elementos observados pelo mecanismo de configuração dinâmica, observações com um número limitado de informações podem não ser suficientes devido às constantes mudanças que sistemas computacionais sofrem em tempo de execução. Observações apuradas precisam coletar quantidades suficientes de dados para definir o comportamento de um elemento e de suas mudanças ao longo do tempo. Por isso, o coordenador também é responsável por armazenar o histórico dos atributos coletados em $zNodes$ baseado em uma janela de eventos.

Assim, pode-se criar uma visão do comportamento do sistema no decorrer do

tempo que permite a definição em tempo real de fatores dificilmente conhecidos *a-priori*, como as condições relacionadas a falhas: tempo médio entre falhas (MTBF), tempo médio entre interrupções (MTTI) e tempo médio para reparo (MTTR) [Egwutuoha et al. 2013]. Além disso, cria-se a oportunidade de observação dos custos relacionados ao procedimento de *checkpoint*, como o tempo gasto em salvamentos e o tempo gasto em operações de recuperação.

Para isso, cada elemento observado possui três $zNodes$ – nodos na estrutura do coordenador – descendentes para armazenar seus dados: um para armazenar o índice do $zNode$ correspondente ao último elemento adicionado ($Last - value$), um nodo com N descendentes para armazenar todos os últimos N valores obtidos por observações ($Values$) e outro para o armazenamento de análises estatísticas ($Analysis$) em seus descendentes. A Figura 2 auxilia no entendimento da organização de nodos referentes ao histórico de um elemento.

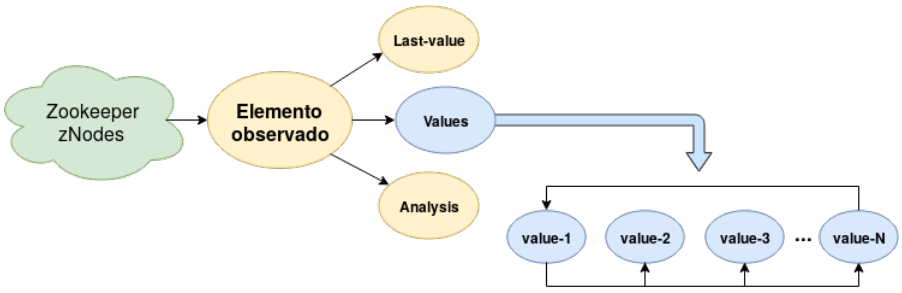


Figura 2. Arquitetura do histórico de atributos com uma janela de N entradas.

4.1. Organização

A organização dos atributos de um elemento no histórico baseia-se em uma janela de observação com tamanho fixo. O nodo $Values$ é composto por um número máximo de N descendentes, sendo que cada um dos N nodos armazena um valor observado. Os nodos são ordenados do menos recente ao mais recente, uma vez que novos valores são adicionados no próximo nodo disponível. Porém, como a janela de observação possui um tamanho fixo, o histórico deve armazenar apenas os últimos N valores observados. O valor de N deve ser especificado no início da execução do coordenador.

A atualização de novos valores quando o histórico armazena o número máximo de entradas é baseada no conceito de *Round Robin Database*, em que o último elemento tem como sucessor o início da lista, para que o ciclo de atualizações não seja interrompido. Deste modo, os valores são atualizados nos descendentes de $Values$ a partir de $value_1$ até $value_N$ (tendo $value_1$ como o primeiro nodo e $value_N$ como o último nodo). Após a atualização de $value_N$, novas entradas de valores observados sobre o mesmo elemento ocorrem a partir de $value_1$ novamente.

O elemento $Last - value$ é essencial para o controle de ordenamento da base de valores, de forma que este nodo armazene o índice do $zNode$ com o atributo coletado mais recentemente. Assim, tem-se uma noção de onde inicia e termina a ordem temporal dos valores armazenados. Essa característica é importante para análises sobre o histórico quando informações mais recentes devem apresentar uma influência diferente das mais

antigas. Além disso, o nodo $Last - value$ serve como suporte para sua observação via $watchers$, já que cada mudança no histórico reflete na mudança do índice atual.

4.2. Análises

A análise dos dados coletados no histórico pode ser armazenada em descendentes do nodo $Analysis$, de modo a criar atalhos para informações já processadas. Neste trabalho, a construção de análises sobre atributos foi feita no elemento supervisor do mecanismo de configuração dinâmica. A partir da modificação de um $zNode$ correspondente ao nodo $Last - value$ de um elemento, o supervisor recebe o alerta do Zookeeper indicando que uma nova análise pode ser realizada com os dados obtidos.

O supervisor, então, estabelece métricas de avaliação dos dados observados e coleta análises de custos. A métrica escolhida pode ser adaptada de acordo com a necessidade de monitoramento. Neste trabalho, o histórico é usado para a coleta de informações sobre o custo – em tempo de execução – de um procedimento de *checkpoint* no Hadoop e também para gerar informações sobre a métrica MTBF do sistema de acordo com um cenário de falhas induzidas estabelecido. Portanto, a análise do supervisor baseou-se na média aritmética simples dos valores observados. Métricas de avaliação mais apuradas para análises complexas serão implementadas em trabalhos futuros.

5. Experimentação

O procedimento de *checkpoint* do Hadoop foi submetido a testes de desempenho para ter seus custos avaliados em diferentes cenários de execução. A análise dos resultados foi dividida no estudo do tempo de execução dos cenários de teste executados, no custo de *checkpoint* e na avaliação do MTBF obtido. Cada cenário de teste foi executado em um total de 20 rodadas e o tempo de execução se deu pela média aritmética das rodadas. A janela de observações do histórico foi configurada com 20 entradas.

5.1. Cenários de Teste

As experimentações foram realizadas na plataforma Grid'5000[1] , um ambiente distribuído de larga escala que fornece uma infraestrutura para testes relacionados a aplicações distribuídas e paralelas [Balouek and et al. 2013]. A configuração usada no *grid* era composta por 8 nós e a configuração de cada nó foi definida com dois processadores Intel Xeon E5-2630v3, 4GB de memória RAM e 200GB de espaço disponível em disco.

Os testes avaliaram execuções do *benchmark* $TestDFSIO$ [Noll 2011], que produz operações de leitura e escrita no HDFS. A escolha deste *benchmark* se deve ao intenso uso de disco, o que permite avaliações em situações de sobrecarga de *I/O*. O $TestDFSIO$ foi configurado para criar (e armazenar no HDFS) arquivos com tamanho de 8GB. Para simular mais de uma situação de sobrecarga, foram definidos cenários com a criação de 24, 32 e 40 arquivos, totalizando cargas de 192GB, 256GB e 320GB respectivamente.

Em relação às configurações de *checkpoint*, foram definidos 4 cenários estáticos usando os seguintes períodos: 3600, 360, 36 e 10 segundos. Ainda, o mecanismo configuração dinâmica foi usado com as aproximação de Young [Young 1974] e de Daly

[1]Grid'5000 é uma plataforma para experimentos apoiada por um grupo de interesses científicos hospedado por Inria e incluindo CNRS, RENATER e diversas Universidades, bem como outras organizações (mais detalhes em https://www.grid5000.fr).

[Daly 2006] para cálculo de períodos ótimos de *checkpoint* baseados no custo de *checkpoints* e na análise de confiabilidade do sistema – com base no tempo médio entre falhas.

Cenários de falhas (CF) também foram definidos para emular situações em que o elemento mestre do HDFS é induzido a uma falha transiente, onde seu processo é encerrado em tempo de execução através do comando $kill$ do Linux e, logo em seguida, seu serviço é reiniciado manualmente. Esta indução caracteriza cenários de falha transiente e permite que a recuperação do HDFS seja avaliada a partir do tempo de execução excedente, adicionado pela carga de leitura do *namespace* salvo pelo *checkpoint* mais recente.

Os CFs foram definidos com três cenários baseados no número de falhas por execução: 1 (CF1), 2 (CF2) e 3 (CF3) falhas. O momento de indução das falhas é escolhido de forma randômica por rodada de execução, com intervalos baseados nos *baselines* sem falhas de cada cenário de teste. A primeira falha ocorre entre 25% e 50%, a segunda entre 50% e 60% e a terceira entre 60% e 75% do *baseline*. No mecanismo dinâmico, o momento de indução é relativo ao *baseline* da configuração estática padrão (3600 seg.).

5.2. Tempo de execução

Os resultados obtidos em relação à execução do *benchmark* são sumarizados na Tabela 1, que exibe o tempo de execução, em segundos, para cada configuração de número de arquivos (NrA), CF e de período estático usado. Além disso, o número de *checkpoints* realizados (NCp) é descrito de acordo com a média de salvamentos por rodada. A Tabela 1 mostra que o período entre *checkpoints* é determinante para a eficiência de recuperação nos diferentes cenários de falha – e em um cenário sem falha – e de carga.

Config.	NrA	**S/falha**	NCp	**CF1**	NCp	**CF2**	NCp	**CF3**	NCp
	24	1524,2	0,20	1715,0	0,48	1743,2	0,56	1807,5	0,47
3600	32	1846,8	0,56	2236,1	0,41	2241,1	0,30	2272,5	0,31
	40	2148,7	0,59	2419,0	0,33	2490,7	0,33	2530,3	0,42
	24	1580,3	4,20	1677,2	4,66	1681,4	3,50	1720,1	3,50
360	32	1998,0	5,41	2061,1	4,73	2154,2	3,75	2168,6	3,55
	40	2281,7	5,33	2389,8	4,33	2394,6	3,83	2482,1	4,09
	24	1611,5	35,60	1649,4	40,01	1668,9	37,56	1677,0	31,00
36	32	2056,2	54,16	2098,5	42,82	2115,6	39,85	2157,6	33,88
	40	2340,5	58,22	2480,9	52,12	2419,2	56,79	2477,6	41,84
	24	1625,4	94,68	1651,0	118,03	1675,5	105,33	1694,5	88,60
10	32	2126,8	178,72	2152,2	175,35	2200,3	141,61	2201,1	109,61
	40	2386,7	198,54	2413,1	189,61	2428,3	181,60	2480,5	179,60

Tabela 1. Tempos de execução (em segundos) e números de *checkpoint* obtidos pelos cenários de testes com períodos estáticos.

Assim como observado em validações anteriores [Cardoso and Barcelos 2018b], o período entre *checkpoints* possui um impacto significativo no tempo de execução das aplicações no Hadoop. Pelos resultados da Tabela 1, nota-se que o estabelecimento frequente de *checkpoints* em tempo de execução proporciona recuperações mais eficientes em relação a um caso de salvamentos a cada 3600 segundos. Ou seja, o tempo gasto para recuperação é menor quando a frequência de salvamentos é maior.

O comportamento da sobrecarga, conforme variou-se o cenário de falhas, mostrou

duas características distintas. Em princípio, a indução de uma única falha (CF1) aumentou consideravelmente o tempo de execução do *benchmark* em todos os cenários de carga. Porém, na segunda e na terceira falhas (cenários CF2 e CF3), a sobrecarga não se mostrou tão relevante. Esse reflexo é observado pois a recuperação do $NameNode$ exige um salvamento de *checkpoint* no momento de reinício do elemento falho. Ou seja, falhas próximas entre si tendem a perder pouco trabalho.

Ainda que a configuração com *checkpoints* mais frequentes possua uma perspectiva de recuperações eficiente, o tempo de execução final em cenários de falha é maior que configurações com 360 segundos e 36 segundos quando apenas uma falha é induzida por execução. Este é um comportamento previsto devido à alta sobrecarga de salvamentos em momentos livres de falha, o que acaba prejudicando o andamento da aplicação independentemente da eficiência de recuperação. Por isso, em cenários de alta confiabilidade do sistema, a alternativa mais viável para a escolha de um período estático deve seguir uma abordagem pouco intrusiva de *checkpoints*.

Por outro lado, aumentando-se o número de falhas simultâneas por execução, configurações com recuperações eficientes tendem a produzir resultados mais satisfatórios. Isto é, a demanda por procedimentos de recuperação é maior com uma quantidade maior de falhas, por isso o estado de atualização do `FSImage` pelo *checkpoint* torna-se essencial nesse caso. Assim, entende-se que alternativas mais cautelosas – com *checkpoints* realizados com maior frequência – são a melhor solução para uma definição estática do período em caso de baixa confiabilidade do sistema.

Com o objetivo de balancear os fatores relacionados ao *checkpoint*, o mecanismo de configuração dinâmica também foi usado. Essa solução apresentou resultados satisfatórios e equilibrados em ambas as variações utilizadas (de carga e de CF). Os resultados, exibidos na Tabela 2, são divididos pela aproximação usada (App) e indicam o tempo de execução para cada cenário de falhas e o número de *checkpoints* realizados (NCp). Como não há informações *a-priori* do MTBF, o cenário sem falhas não foi executado para o mecanismo dinâmico. Nesse caso, a falta de informação sobre a quantidade de falhas impossibilita o uso das fórmulas de Young e Daly.

App	NrA	**CF1**	NCp	**CF2**	NCp	**CF3**	NCp
Young	24	1640,7	5,08	1670,6	5,45	1738,9	6,43
	32	2066,8	7,56	2093,0	8,43	2138,2	9,40
	40	2345,4	8,40	2363,1	9,57	2398,8	10,60
Daly	24	1654,6	4,23	1679,7	7,55	1738,9	7,68
	32	2064,9	6,78	2129,5	7,62	2151,3	8,53
	40	2339,5	7,67	2384,1	6,89	2419,2	8,83

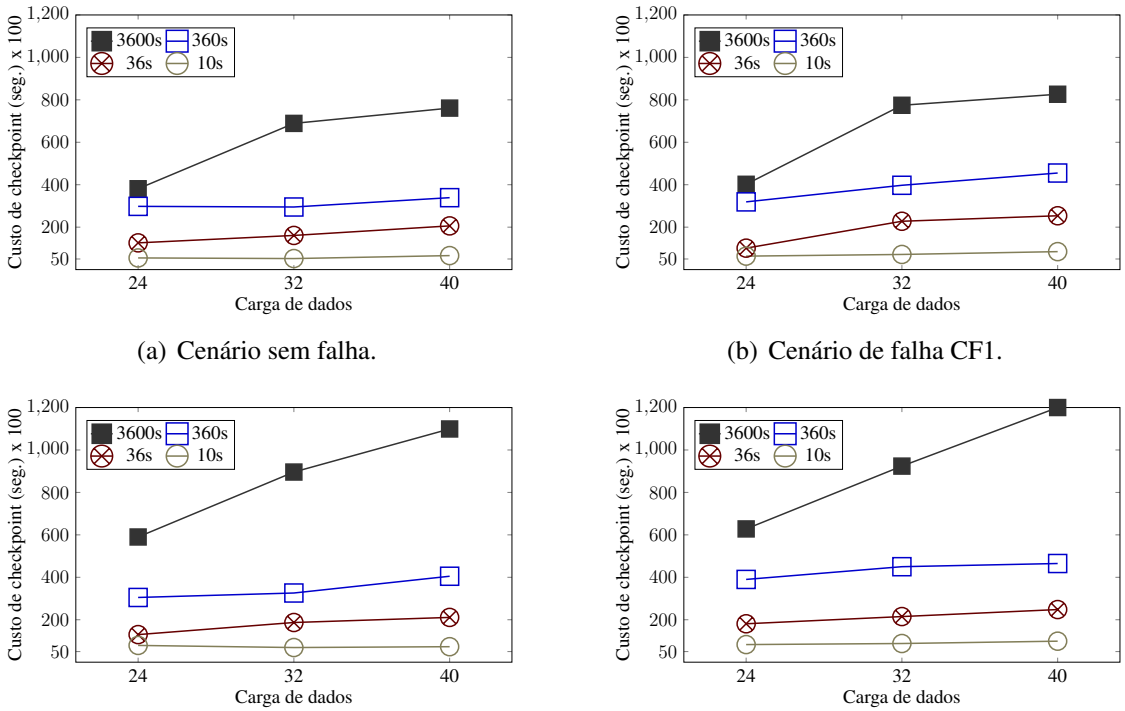
Tabela 2. Tempos de execução (em segundos) e números de *checkpoint* obtidos pelos cenários de testes com configuração dinâmica.

Os dois fatores usados nas métricas de Young e Daly são o custo de *checkpoint* (ligado à intrusividade de salvamentos em momentos livres de falha) e o MTBF (ligado à eficiência de recuperação). Por consequência, adaptações baseadas no histórico conseguem equilibrar com sucesso o período de acordo com comportamentos sempre atuais do sistema. Assim, a aplicação do mecanismo de configuração dinâmica auxilia em casos de indecisão ou desconhecimento do comportamento do sistema.

Situações de baixa confiabilidade, em que a ocorrência de falhas é frequente, podem ser identificadas quando as aproximações de Young e Daly são usadas, devido ao fator MTBF. Já os cenários de intrusividade também são identificados e amenizados pelas aproximações, já que o custo do procedimento também é considerado. Ao contrário dos casos de configurações estáticas, o número de *checkpoints* realizados é maior com mais falhas quando o período é dinamicamente configurado.

5.3. Custo de *checkpoint*

O custo de *checkpoint* é calculado pelo tempo de execução necessário para que o NN atualize a imagem de *checkpoint*, desde o envio do *log* de edições ao SNN até o recebimento da imagem atualizada. A Figura 3 apresenta uma análise referente ao custo estimado da realização do procedimento de *checkpoint* nos diferentes cenários de teste. A estimativa é feita pelo histórico do mecanismo dinâmico.



(a) Cenário sem falha.

(b) Cenário de falha CF1.

(c) Cenário de falha CF2.

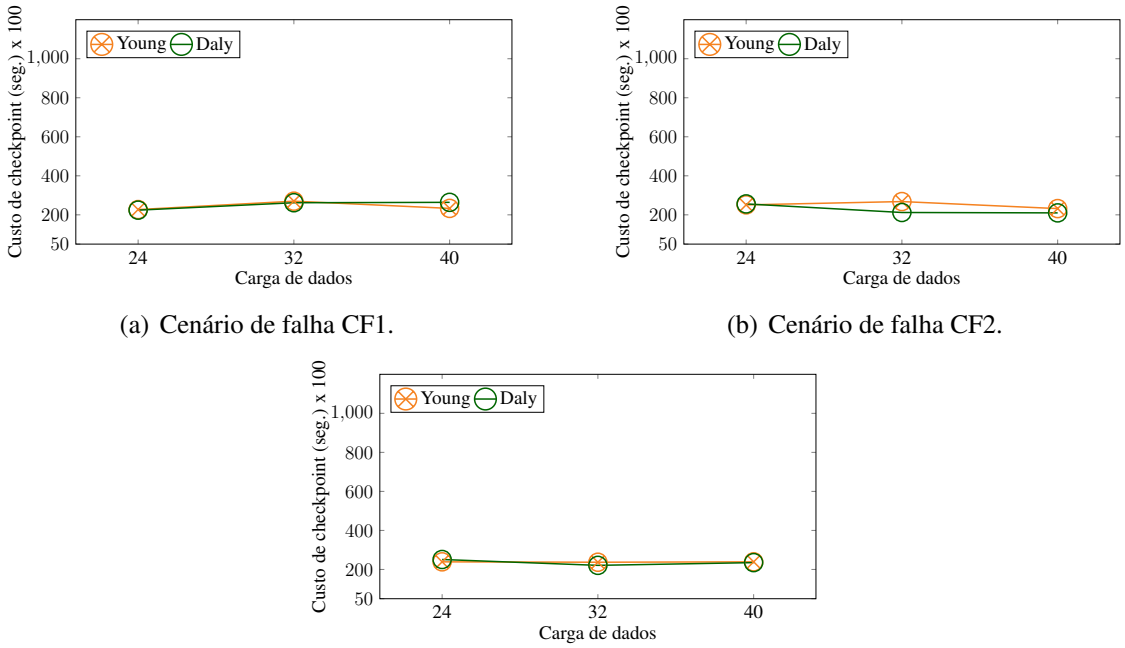
(d) Cenário de falha CF3.

Figura 3. Custo de *checkpoint* com períodos estaticamente configurados.

Com configurações estáticas, é evidente que o custo diminui conforme a frequência de salvamentos aumenta. Com um tempo menor entre salvamentos, há menos operações no `EditLog` e, consequentemente, menos trabalho para atualizar-se o `FSImage`. Variando-se os cenários de falha, os valores de custo para uma mesma frequência de *checkpoint* não possuem grandes diferenças. Com exceção da configuração estática padrão (3600 segundos), há uma tendência de custos maiores com mais falhas. A Tabela 1 auxilia na compreensão desse comportamento: uma vez que cenários com mais falhas diminuem o número de *checkpoints*, o custo médio de salvamentos aumenta.

Nos cenários de aplicação do mecanismo dinâmico – conforme gráficos da Figura 4 –, os resultados obtidos se mostraram equilibrados, com pouca diferença entre as

aproximações usadas. A média do custo de salvamento para ambas as fórmulas se manteve entre 20000 e 25000 segundos, mesmo com variações de carga e de CF. É importante ressaltar que o mecanismo dinâmico não foi executado em cenários sem falha, devido à falta de informações *a-priori* do MTBF.



(a) Cenário de falha CF1.

(b) Cenário de falha CF2.

(c) Cenário de falha CF3.

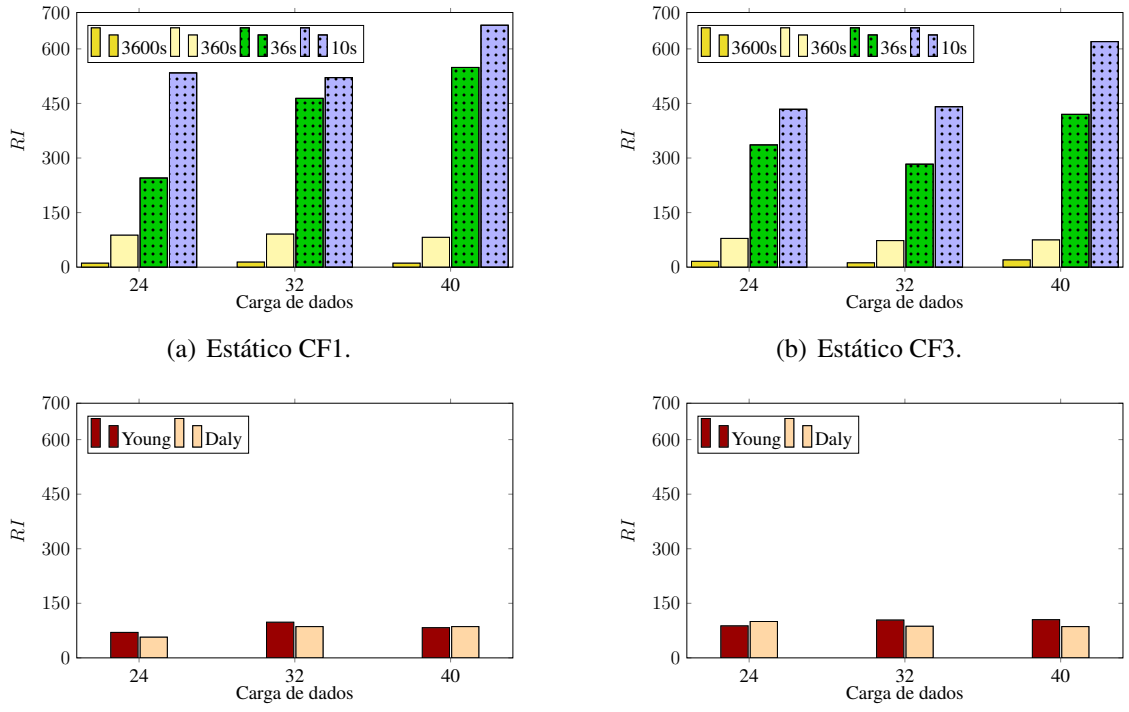
Figura 4. Custo de *checkpoint* com períodos dinamicamente configurados.

Mesmo que a diferença da variação de custo no mecanismo dinâmico (para uma mesma carga de dados) seja menos evidente que no cenário estático, uma quantidade maior de falhas minimiza os custos. Com falhas mais frequentes, o MTBF estimado tende a ser menor e, por isso, as aproximações de Young e Daly inclinam-se para a escolha de períodos mais curtos. Esse resultado demonstra como o mecanismo dinâmico consegue tempos de execução satisfatórios mesmo com a indução de falhas: o custo de *checkpoints* tende a ser equilibrado para que a recuperação e a intrusividade não sejam custosas.

5.4. Relação de impacto

Para que a eficiência da técnica de *checkpoint* seja avaliada, é importante definir o impacto da configuração adotada em relação aos custos envolvidos. Os gráficos da Figura 5 exibem as relações de impacto (RI) obtidas de acordo com a fórmula: $RI = (Custo * NrC)/TE$. Esse valor calcula a fração de tempo gasto com operações de *checkpoint* – dado pelo custo médio para um salvamento ($Custo$) multiplicado pelo número médio de *checkpoints* (NrC) – e o coloca em razão do tempo de execução (TE).

O objetivo da fórmula da RI é verificar o impacto dos salvamentos de *checkpoint* no desempenho do *benchmark*. Os resultados da Figura 5 mostram que custos menores obtidos por períodos estáticos e frequentes de *checkpoint* não compensam suas utilizações em momentos livres de falha. A relação de tempo gasto com *checkpoints* a cada 10 segundos é expressivamente maior que os outros casos de períodos estáticos. Já uma



(a) Estático CF1.

(b) Estático CF3.

(c) Dinâmico CF1.

(d) Dinâmico CF3.

Figura 5. Relação de custo e número de *checkpoints* por execução de acordo com o tempo de execução.

menor frequência de salvamentos gera menos intrusividade ao sistema. Desta maneira, mesmo que o custo médio de um único *checkpoint* seja maior, a quantidade reduzida de salvamentos torna a relação de tempo gasto menor.

Quando os resultados das Tabelas 1 e 2 são relacionados com a Figura 5, é possível notar que um período de *checkpoint* estático pode ser eficiente em uma ocasião específica, mas compromete o desempenho do *benchmark* conforme o cenário de execução muda. Neste caso, períodos dinamicamente configurados se mostraram "adaptados" a mudanças de carga e de cenário de falha. Ou seja, ambas as aproximações usadas apresentaram desempenhos satisfatórios para todos os cenários testados.

Comparando-se as Figuras 5(a) e 5(b) (RI dos cenários estáticos) com as Figuras 5(c) e 5(d) (RI dos cenários dinâmicos), pode-se observar com nitidez a diferença do impacto do *checkpoint* em cada mecanismo de configuração. Enquanto a abordagem estática alcança valores altos de RI, o mecanismo dinâmico atenua o impacto do *checkpoint* e mantém os custos equilibrados com o tempo de execução, satisfazendo o propósito das aproximações [Young 1974] [Daly 2006].

6. Considerações finais

Este trabalho explorou o impacto de variações na técnica de *checkpoint* do Apache Hadoop em relação aos custos e tempos de execução frente a cenários de teste. Os experimentos realizados evidenciaram a necessidade do uso do mecanismo de configuração dinâmica para a definição do período entre *checkpoints*. Com adaptações em tempo real, os períodos são equilibrados para que as vantagens de cada caso sejam exploradas.

As vantagens do mecanismo de configuração dinâmica ficam evidentes com o estudo do impacto de salvamentos sobre o tempo de execução em diferentes cenários de teste. Casos com períodos estaticamente configurados apresentaram diferentes resultados. Já períodos dinamicamente configurados mostraram estabilidade de custos e número de *checkpoints*, destacando os resultados satisfatórios obtidos nos cenários testados.

Em próximos passos, a análise de custos será ampliada para que se obtenham melhores desempenhos no mecanismo de configuração dinâmica, com desenvolvimento de novas aproximações para a definição de períodos ideais. Também, validações serão estendidas para uma maior variedade de cenários execução. Por fim, o mecanismo de configuração dinâmica será explorado em outras ferramentas, como o Apache Spark.

Referências

Balouek, D. and et al. (2013). Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*. Springer Intl Publishing.

Cardoso, P. V. and Barcelos, P. P. (2018a). Experimentação e análise de checkpoint dinâmico no apache hadoop sob cenários de falha. In *XIX Simpósio de Sistemas Computacionais de Alto Desempenho (WSCAD 2018)*. No prelo.

Cardoso, P. V. and Barcelos, P. P. (2018b). Validation of a dynamic checkpoint mechanism for apache hadoop with failure scenarios. In *Test Symposium (LATS), 2018 IEEE 19th Latin-American*, pages 1–6. IEEE.

Cui, L., Hao, Z., Li, L., Fei, H., Ding, Z., Li, B., and Liu, P. (2015). Lightweight virtual machine checkpoint and rollback for long-running applications. In *Int. Conference on Algorithms and Architectures for Parallel Processing*, pages 577–596. Springer.

Daly, J. T. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems*, 22(3):303–312.

Egwutuoha, I. P., 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.

Ghit, B. and Epema, D. (2017). Better safe than sorry: Grappling with failures of in-memory data analytics frameworks. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM.

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, page 9.

Laprie, J.-C. (1985). Dependable computing and fault tolerance: Concepts and terminology. In *25th International Symposium on Fault-Tolerant Computing, 1995*. IEEE.

Noll, M. (2011). Benchmarking and stress testing an hadoop cluster with terasort, testdfsio & co. *Online: http://www. michael-noll. com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoopcluster-with-terasort-testdfsio-nnbench-mrbench*.

White, T. (2015). *Hadoop: The Definitive Guide, 4th Edition*. "O'Reilly Media, Inc.".

Young, J. W. (1974). A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531.