Arquitetura para Elaboração de um Histórico de Atributos Relacionados à Técnica de Checkpoint and Recovery

Paulo V.M. Cardoso[1], Rhauani Weber Aita Fazul[2], Patr´ıcia Pitthan Barcelos[1]

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

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

Abstract. *Optimal attributes for checkpoint and recovery technique are hard to find due to system distinct behaviours over the time. This work proposes an architecture to store system elements usage on a historical record structure. The goal of historic is to provide statistical informations about system usage, in order to help on checkpoint period taking decisions. In this way, the architecture already proposed for dynamic checkpoint configuration will be able to use approximations for setting proper checkpoint attributes.*

1. Introdução

A técnica de *Checkpoint and Recovery* (CR) é amplamente usada como uma solução tolerante a falhas, devido a sua eficiência em recuperações de falhas em sistemas computacionais [Egwutuoha et al. 2013]. Seu funcionamento consiste em duas fases: o estabelecimento de *checkpoints* e a recuperação, que acontece após uma falha e consiste em recuperar o andamento normal do sistema a partir do seu estado estável mais recente.

Porém, a configuração de atributos relacionados ao *checkpoint* pode ser um grande desafio. Ferramentas de processamento distribu´ıdo, como o Apache Hadoop e o Apache Spark, foram criadas para suportar aplicações com grandes quantidades de dados. Os dois *frameworks* possuem suas variações da técnica de *checkpoint* implementadas para garantir confiabilidade de execução. Porém, a eficiência dessas implementações depende de estabelecimentos eficientes de *checkpoints*, para que a confiabilidade não prejudique o desempenho do sistema e das aplicações.

Nas versões padrão do Hadoop, a determinação do período entre *checkpoints* é essencial para o desempenho e para a confiabilidade do sistema. O estabelecimento frequente de *checkpoints* acelera o procedimento de recuperação pós-falha, mas adiciona intrusividade pela grande quantidade de operações de *I/O* envolvida. Enquanto isso, períodos mais longos entre *checkpoints* tornam a recuperação mais lenta ao mesmo tempo em que diminuem a intrusividade.

Já no Spark, as políticas de seleção de *datasets* para *checkpoint* também interferem no comportamento da ferramenta. Neste caso, um *checkpoint* elimina a linha do tempo de um *dataset* e o salva em um repositório seguro. Em caso de falha, a recuperação inicia a partir do *dataset* salvo em *checkpoint*, tornando sua reconstrução mais rápida. Porém, o estabelecimento ineficiente de *checkpoints* também adiciona intrusividade ao sistema. Como a política de seleção depende do desenvolvedor, escolhas inapropriadas tendem a prejudicar o desempenho do sistema e o uso de *checkpoints* perde seu propósito.

Em ambos os sistemas, alterações nos atributos da técnica de CR em tempo real são inviáveis. Isto é, deve-se interromper todos os serviços ativos para que uma modificação de atributo (ou de política) tenha efeito. Uma solução para este problema é o uso de um mecanismo de configuração dinâmica para atributos de *checkpoint* baseado em métricas de monitoramento, através das variações do Hadoop e do Spark propostas em [Cardoso and Barcelos 2018].

A arquitetura para configuração dinâmica dos atributos de *checkpoint* possui dois elementos principais: o supervisor, capaz de identificar o comportamento dos nodos e de fatores do sistema em tempo real, e o coordenador que fornece uma interface entre os *frameworks* e o supervisor. Porém, a definição das métricas de monitoramento pode se tornar uma tarefa complexa à medida que informações sobre o ambiente computacional são desconhecidas. Aproximações para períodos ideais requerem um conhecimento *a-priori* de fatores do sistema que geralmente são desconhecidos do desenvolvedor.

2. Solução Proposta

Para auxiliar na definição das métricas de monitoramento, este trabalho propõe e define uma arquitetura para o armazenamento do histórico de utilização de elementos do ambiente computacional. O histórico é criado em uma estrutura de árvore, onde seus nodos representam os elementos a serem monitorados. Assim, tem-se uma visão do comportamento do sistema no decorrer do tempo, a partir de fatores relacionados a falhas como o tempo médio entre falhas (*Mean Time Between Failures*, ou MTBF) [Egwutuoha et al. 2013]. Também pode-se observar os custos do procedimento de *checkpoint*, além do uso de elementos de *hardware*, como CPU, disco e memória RAM.

O objetivo da manutenção de um histórico de atributos é possibilitar uma análise de características do sistema que seriam inviáveis em uma abordagem *a-priori*. Desta forma, a aplicação de aproximações para a definição de atributos ideais de *checkpoint* é facilitada e pode ser anexada à arquitetura de configuração dinâmica. Com informações atualizadas sobre o comportamento do sistema, a aplicação dessas soluções pode auxiliar na tomada de decisões sobre modificações em tempo real dos atributos de *checkpoints*.

O histórico foi desenvolvido com o auxílio da ferramenta 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 nodos chamados *zNodes*.

Para cada elemento monitorado, um *zNode* pode ser definido com um caminho específico na estrutura de árvore do Zookeeper. Nessa estrutura, um elemento possui três nodos descendentes: um para armazenar o último elemento adicionado (*last*), um nodo para armazenar os valores obtidos por observações (*values*) e outro para o armazenamento de análises estatísticas (*analysis*).

Para que se obtenha uma visão a respeito do comportamento de um elemento, é necessário armazenar uma quantidade suficiente de informações. Por isso, o nodo *values* é composto por um número máximo de N descendentes, que corresponde à janela de observação. Logo, o histórico armazena os últimos N valores observados. A atualização é baseada no conceito de *Round Robin Database* (RRD), de modo que os valores são atualizados nos descendentes de *value* a partir de 1 até N e, após N, a partir de 1 novamente.

Uma importante função do Zookeeper é o mecanismo de *watcher*, que permite o envio de alertas para qualquer sistema que observe um *zNode* assim que este nodo é modificado. Nesse caso, o nodo *last* é usado para observação externa, de modo que qualquer ferramenta tenha a noção de quando um elemento tem seus dados atualizados. O elemento *last* é essencial para o controle de ordenamento da base de valores, de modo que este nodo armazena 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* serve como suporte para sua observação via *watchers*, já que cada mudança no histórico reflete na mudança do índice atual.

A análise dos dados coletados no histórico pode ser armazenada em descendentes do nodo *analysis*, de modo a criar atalhos para informações já processadas. No contexto deste trabalho, a construção de análises sobre atributos é feita no elemento supervisor do mecanismo de configuração dinâmica. A partir da modificação de um *zNode* correspondente ao nodo *last* de um elemento, o supervisor recebe o alerta do Zookeeper indicando que uma nova análise pode ser realizada com os dados obtidos.

O supervisor 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, três abordagens foram definidas. A média aritmética dos valores armazenados descreve uma abordagem simples. Já a média ponderada pode ser usada para valorizar observações mais recentes. Análises mais apuradas podem envolver uma solução baseada em modelos auto-regressivos (*autoregressive-moving-average*).

3. Considerações Finais

Este trabalho apresentou uma abordagem para o armazenamento das estatísticas de uso de elementos em um sistema computacional, baseado em um histórico de observações. Essa arquitetura pode ser útil para a definição de fatores de utilização do sistema que seriam inviáveis em observações a-priori. Nos próximos passos, o histórico será validado junto ao mecanismo de configuração dinâmica de *checkpoint*, de modo a encontrar soluções ideais para a técnica de acordo com a estatística de uso dos elementos observados.

Referências

Cardoso, P. V. and Barcelos, P. P. (2018). Dynamic checkpoint architecture for reliability improvement on distributed frameworks. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE.

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.

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.