Monitoramento Computacional com o Apache ZooKeeper

Rafael de Lima Trindade Silva[1], **Raíssa Arantes**[1],
Rhauani Weber Aita Fazul[2], **Patrícia Pitthan Barcelos**[2]

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

{rdsilva, rarantes, rwfazul, pitthan}@inf.ufsm.br

Resumo. *O Apache ZooKeeper é amplamente utilizado para a coordenação de sistemas distribuídos. Sua estrutura hierárquica e centralizada de dados permite que mudanças sejam gerenciadas de maneira eficiente. Neste trabalho, o ZooKeeper foi usado para a implementação de um sistema de monitoramento, que baseia-se na coleta de informações do ambiente. Com isso, é possível organizar os dados de monitoramento e notificar sobre mudanças no sistema.*

1. Introdução

O monitoramento do desempenho de um sistema é importante para conceder informações sobre seu estado de funcionamento. Essas informações podem ser utilizadas para a detecção de possíveis falhas e problemas futuros, além de viabilizarem otimizações de desempenho por meio de ações realizadas a partir dos dados coletados. Nesse sentido, o *framework* Apache ZooKeeper [Foundation 2021] permite, por meio de seu *namespace* compartilhado, armazenar informações de coordenação e gerar notificações de mudanças.

Este trabalho apresenta um sistema de monitoramento baseado nas funcionalidades disponibilizadas pelo ZooKeeper, onde o *framework* foi utilizado para armazenar os dados de monitoramento coletados. Além da organização dos dados relacionados aos recursos do sistema, alertas constantes sobre o estado do ambiente são gerados através de mecanismos integrados ao serviço do ZooKeeper. Um experimento prático foi realizado a fim de testar o sistema de monitoramento proposto e seu mecanismo de notificações aos processos destinados à tomada de ações corretivas.

O trabalho está organizado em cinco seções. A Seção 2 apresenta o *framework* Apache ZooKeeper. A Seção 3 elenca os principais trabalhos relacionados. A Seção 4 detalha a metodologia aplicada no desenvolvimento do trabalho. A Seção 5 exibe os resultados obtidos. Por fim, a Seção 6 conclui o artigo e aponta os trabalhos futuros.

2. Apache ZooKeeper

O Apache ZooKeeper[1] é um projeto que fornece soluções de código aberto para diversos problemas de coordenação em sistemas distribuídos [Haloi 2015]. A ferramenta foi projetada para executar como um serviço centralizado, escalável e altamente confiável.

Seguindo uma estrutura similar a de um sistema de arquivos, o ZooKeeper possui um modelo de armazenamento hierárquico de dados baseado em um repositório central. Todavia, ao contrário de um sistema de arquivos convencional, que é projetado para o

[1]https://zookeeper.apache.org/

armazenamento persistente, os dados do ZooKeeper são mantidos em memória, possibilitando que as operação sobre seu *namespace* compartilhado tenham alto desempenho e baixos níveis de latência, além da garantia de durabilidade e atomicidade [Foundation 2021]. Os dados no ZooKeeper são organizados sob a forma de uma árvore de registradores de dados, denominados *znodes* na nomenclatura do ZooKeeper [Junqueira and Reed 2013].

Para possibilitar alto desempenho e escalabilidade, o ZooKeeper necessita de estratégias eficientes para o atendimento de requisições. Realizar o acesso a um *znode* sempre que for necessário verificar se seu conteúdo foi alterado (*polling*), pode causar um custo elevado e impossibilitar o alto desempenho esperado do *framework* [Haloi 2015]. Pensando nisso, o ZooKeeper dispõe de um poderoso mecanismo de notificações que permite observar mudanças na árvore de *znodes*: os *watches* [Foundation 2021]. Um *watch* pode ser setado, pelo cliente, em um *znode* específico. Assim, no caso de mudanças terem ocorrido no conteúdo do *znode* monitorado ou em sua sub-árvore, o cliente é notificado.

Desta forma, com a estrutura flexível dos *znodes*, é possível armazenar, em tempo de execução e de forma eficiente, informações sobre o ambiente computacional. Com os *watches*, por sua vez, permite-se notificar o usuário sobre alterações nos parâmetros monitorados ou quando esses atingirem determinados níveis e exigirem a execução de alguma ação corretiva, tal como encerrar um processo que esteja sobrecarregando o sistema.

3. Trabalhos Relacionados

O monitoramento de ambientes computacionais é objeto de estudo de diferentes trabalhos na literatura. Dentre elas, o Zabbix [Olups 2016], uma ferramenta de código aberto que possui mecanismos de alerta – por e-mail, SMS, dentre outros –, além de fornecer monitoramento de recursos e relatórios de dados. Outra ferramenta, o Cacti [Kundu and Lavlu 2009], é voltada para coleta de dados de elementos de redes de computadores, como a largura de banda sendo utilizada, CPU, e outros parâmetros, permitindo também um monitoramento gráfico do ambiente computacional.

4. Metodologia

O monitoramento de desempenho do sistema operacional foi desenvolvido e executado em um processador dual *core* no ambiente *macOS*, utilizando o ZooKeeper (versão 3.6.2) com a linguagem de programação *Python 3*. Para essa linguagem, a biblioteca *Kazoo*[2], que permite a integração com o ZooKeeper de forma mais compreensível para o desenvolvedor, foi utilizada durante o desenvolvimento.

Os parâmetros selecionados para o monitoramento foram: (i) a quantidade de memória RAM em uso; (ii) a utilização da CPU; (iii) o uso do disco (HDD); e (iv) o número de processos em execução no sistema. Com base nesses parâmetros, é possível monitorar o desempenho do sistema e, futuramente, exercer medidas corretivas para melhorá-lo. Neste trabalho, cada um dos parâmetros monitorados possui um *znode* associado na árvore do ZooKeeper, de forma que qualquer cliente que tenha um *watch* nesse *znode* seja notificado sobre possíveis mudanças.

[2]`https://kazoo.readthedocs.io/`

4.1. Desenvolvimento do Sistema de Monitoramento

Para a implementação do sistema de monitoramento, primeiramente, foi realizada a criação da árvore de *znodes*, estruturada com um *znode* para cada parâmetro analisado. A etapa seguinte foi a implementação dos *watches*, um para cada *znode*. Comumente, os *watches* precisam ser registrados a cada ocorrência de um evento no *znode* ao qual esta está vinculada [Junqueira and Reed 2013]. Entretanto, com a biblioteca *Kazoo*, isso não é necessário, uma vez que ela possibilita o uso de uma API que dispensa a reconfiguração recorrente. Assim, uma vez estabelecido, o *watch* continuará a notificar o cliente enquanto a conexão estiver aberta. Essa funcionalidade é obtida através do método *DataWatch*.

Na sequência à criação dos *watches*, criou-se a função (*handler*) a ser executada toda vez em que uma modificação for percebida na árvore de *znodes*. A seguir, foi criado um *script Python* responsável pela coleta das informações na máquina. Essas informações, posteriormente, são atualizadas na árvore de *znodes* e registradas em *logs*, visando manter o histórico dos dados buscados.

4.2. Experimentação

Para a execução do sistema de monitoramento, é necessário um servidor ZooKeeper em execução. Neste trabalho foi realizado um monitoramento local, uma vez que buscou-se experimentar os recursos do Zookeeper em um sistema de monitoramento pequeno e controlado para, futuramente, aplicá-lo em um sistema de maior escala. Para tanto, o Zookeeper foi executado em modo *standalone*.

Após, executou-se o *script Python* responsável pela coleta e armazenamento das informações. Foram realizadas 20 execuções com o *NovaBench*[3], um *benchmark* voltado para testes de desempenho do sistema computacional. Este *benchmark* foi utilizado com o objetivo de gerar carga durante a coleta dos dados, sendo testados parâmetros como a CPU, GPU, RAM e disco, forçando assim, a capacidade do sistema. Em cada execução, os parâmetros foram coletados 17 vezes, com um intervalo de 5 segundos entre cada coleta. Dessa forma, foi possível analisar as mudanças que ocorreram no contexto do sistema e o disparo dos *watches* quando alterações significativas eram identificadas.

5. Resultados e Discussão

A Tabela 1 exibe a média aritmética das 17 coletas dos parâmetros, que foram realizadas em cada uma das 20 execuções. Ao início de cada execução, não havia nenhuma aplicação em andamento no sistema, o que é reforçado ao analisar a coleta 1, que apresenta um baixo uso de CPU. Após a coleta 1, quando é iniciada a execução do *NovaBench*, o uso de CPU teve aumento significativo, chegando a 100% em algumas das execuções. Esse aumento, por sua vez, dispara os *watches*, que exibem um aviso ao usuário, indicando que houve um uso elevado de CPU. Percebe-se também que, a partir da coleta 13, que corresponde ao final da execução do *benchmark*, há uma queda considerável no uso de CPU. Em relação ao uso de disco não houve variações durante as execuções (desvio padrão de 0%).

Em relação ao número de processos ativos, não foram identificadas mudanças significativas. O uso da memória também teve pouco impacto com a execução do *benchmark* no sistema. Todas as informações coletadas são automaticamente salvas em *logs*, assim

[3] `https://novabench.com`

Tabela 1. Média dos parâmetros coletados durante as execuções.

Coleta	Uso de CPU (%)	Número de Processos	Uso de Memória (%)	Uso de Disco (%)
1	15,57	432	54,43	19,49
2	94,50	434	54,55	19,49
3	31,20	434	54,57	19,49
4	95,43	434	54,55	19,49
5	30,63	434	54,51	19,49
6	28,02	433	54,49	19,49
7	95,17	433	56,57	19,49
8	43,60	433	56,00	19,49
9	36,90	433	63,53	19,49
10	40,06	433	63,51	19,49
11	43,52	432	63,54	19,49
12	42,50	432	63,41	19,49
13	12,02	433	56,02	19,49
14	5,20	433	55,56	19,49
15	5,67	434	54,59	19,49
16	4,62	433	54,39	19,49
17	4,80	433	54,27	19,49
σ	31,17	0,70	3,79	0

permitindo uma posterior análise da experimentação. Isso fornece subsídios para realizar previsões do estado do ambiente e, assim, conduzir ações visando minimizar os efeitos de possíveis interferências de desempenho no sistema. Considerando o uso de CPU que chegou próximo a 100% em algumas coletas, uma possível intervenção após o disparo do *watch* é verificar a relevância dos processos em execução no sistema a partir dos dados coletados. Assim, se aplicável, pode-se suspender o processo que está com alto consumo de CPU ou então tomar uma ação corretiva de maior impacto, como o encerramento de um processo não essencial que esteja consumindo muitos recursos do sistema.

6. Considerações Finais

Este trabalhou se destinou ao desenvolvimento de um sistema de monitoramento do ambiente computacional com o Apache ZooKeeper. Através de sua estrutura de armazenamento baseada em *znodes* e em *watches*, foi possível organizar os dados coletados e gerar notificações assim que uma mudança de contexto for observada no sistema.

Trabalhos futuros envolvem a validação em um ambiente distribuído e a criação de uma estrutura para a tomada de ações corretivas mais efetivas, baseando-se nos dados de *logs* e nas informações de estado mantidas no ZooKeeper. A esse respeito, os *watches* são recursos que podem auxiliar na aplicação do sistema de monitoramento em sistemas de maior escala, uma vez que permitem gerar notificações de mudanças sobre os parâmetros de desempenho em cada nodo para posterior ação corretiva.

Referências

Foundation, A. S. (2021). "ZooKeeper". `https://zookeeper.apache.org/doc/r3.6.1/zookeeperOver.html`. Janeiro.

Haloi, S. (2015). *Apache Zookeeper Essentials*. Packt Publishing Ltd, 1st edition.

Junqueira, F. and Reed, B. (2013). *ZooKeeper: distributed process coordination*. "O'Reilly Media, Inc.".

Kundu, D. and Lavlu, S. I. (2009). *Cacti 0.8 network monitoring*. Packt Publishing Ltd.

Olups, R. (2016). *Zabbix Network Monitoring*. Packt Publishing Ltd.