Banco de dados de chave-valor distribuído YEK

Silvana Trindade, Maurício André Cinelli

22 de novembro de 2015

Introdução

Antes de entender como o YEK funciona, é importante ressaltar porquê ele funciona da maneira que funciona. Aplicações como o YEK, um banco de dados chave-valor, geralmente são utilizadas por outras aplicações como serviços de cache, armazenamento de sessões, e dados propriamente ditos. Sendo serviços importantes, é necessário que estes dados tenham alta disponibilidade. Claramente, um servidor não é suficiente pra garantir que estes dados se mantenham disponíveis uma vez que este pode, por diversos motivos, *cair*.

É neste ponto que o YEK entra, seu ponto forte é a capacidade de ser distribuído em milhares de servidores, garantindo a disponibilidade dos dados, mas também economia de recursos. Isto é conhecido como crescimento horizontal, visto que o serviço pode ser composto por milhares de máquinas de baixo custo, sendo que a Google faz uso extensivo disto.

Requisitos

O YEK foi testado apenas em ambiente Linux até o momento. Há uma chance de o mesmo funcionar em Mac OSX, porém nada garantido.

Além disso, se faz necessário que os seguintes pacotes estejam instalados:

- Java SDK 8
- Make

Como compilar

O YEK possui redistribuição já contido em JARs, evitando a necessidade de compilar o projeto.

Porém, caso deseje compilar o código, baixe o código através do git ou um zip diretamente no Github. Com um terminal, acesse a pasta contendo o código descompactado, e execute o seguinte comando:

duiosauods

Como instanciar nós da rede

No YEK há duas formas de instanciar nós na rede. Uma delas é especial para o primeiro nó da rede, já que ele possui configurações garantindo que outro nó possa se conectar à rede.

Para instanciar o primeiro nó, use o comando a seguir:

java -jar yek.jar

Este comando irá instanciar o nó inicial na rede, utilizando a porta 32000 fixa. Com isso podemos instanciar outros nós na rede através do comando a seguir:

java -jar yek.jar IP_ADDRESS PORT

Neste caso, substitua *IP_ADDRESS* pelo IP onde o nó que irá se conectar está rodando, e *PORT* pela porta do nó a se conectar (no caso do nó incial, 32000). Execute este comando quantas vezes desejar, na mesma máquina ou não (preferivelmente que seja em máquinas distintas).

Para testar se os servidores estão funcionando, é possível utilizar um cliente para se conectar à rede e enviar requisições, utilizando o comando a seguir

java -jar yek_cli.jar IP_ADDRESS PORT

O software cliente irá disponibilizar uma interface para realizar operações na rede, tais como inserção, atualização, remoção e busca de dados.

Testes

Infelizmente, o YEK ainda não possui testes automatizados para garantir seu funcionamento, mas pode-se testar seu funcionamento correto através do cliente terminal.

Nos testes realizados verificamos o seguinte:

- os nós levam cerca de 4s para se estabilizar na rede
- ao cair o predecessor, leva cerca de 8-10s para assumir responsabilidade sobre os dados
- pode cair um nó por vez, dando o tempo da rede se reorganizar

- ao entrar na rede, leva cerca de 8s para receber os dados e cópias que devem ser deste nó

A seguir, apresentamos uma descrição mais detalhada sobre os aspectos por trás do YEK e de como ele funciona.

Descrição

Existem vários modelos de armazenamento de dados noSQL (*Not Only SQL*), como por exemplo: chave-valor, documento, colunas, e grafos. O modelo chave-valor, apresenta a proposta que permite que o desenvolvedor efetue a persistência de dados totalmente livre de definições de estrutura para esquema [4], este trabalho terá foco neste modelo. No modelo orientado a grafo, o banco de dados é visto como um multigrafo rotulado e direcionado, onde cada par de nós pode ser conectado por mais de uma aresta.

Em sua essência o orientado a documentos é semelhante ao chave-valor [4]. Entretanto em vez de armazenar qualquer arquivo binário como valor de uma chave é requisitado que o valor dos dados que serão armazenados possua um formato que o banco possa interpretar. No modelo orientado a colunas os dados são indexados por uma tripla (linha, coluna e timestamp), onde linhas e colunas são identificadas por chaves e o timestamp permite diferenciar múltiplas versões de um mesmo dado [5].

Quando se trata de alta performance e disponibilidade em softwares, geralmente utiliza-se apenas acesso em dados que sejam armazenados por chave-valor, ou seja, armazenam valores quaisquer indexados por chaves únicas. Isto torna possível grande escalabilidade horizontal, pois como não é um banco de dados relacional, não há ligação entre os dados, tornando-os independentes, podendo assim serem facilmente divididos em servidores dentro de um *cluster*. Em sistemas relacionais tradicionais, tratar escalabilidade de leitura e escrita é complexo, se não impossível de atingir, devido à forma que os dados são armazenados – um problema que em sistemas noSQL não ocorre.

O sistema chave-valor pode ser visto como uma tabela *hash* distribuída em grande escala, persistente e tolerante à falhas [1]. É importante destacar que este modelo de armazenamento não é suficiente e recomendado para todos os problemas de armazenamento, possuindo seus prós e contras: não é possível fazer consultas com filtros

avançados; não possui chave estrangeira; *joins* precisam ser feitos em código.

Este modelo, porém, torna a performance previsível; é facilmente distribuída entre servidores; ajuda a separar dados e lógica; pode ser usado como mecanismo de cache, e muitos dos conceitos aqui geralmente são empregados em software quando a performance se torna necessária.

Uma característica de sistemas noSQL é que estes não garantem as propriedades conhecidas por ACID (A - atomicidade, C - consistência, I - isolamento, D - durabilidade), porém introduzem outro problema, conhecido por CAP (C - consistência, A - disponibilidade e P - tolerância à partição). Sistemas chave-valor tendem a ser *AP* ou *CP*.

O sistema a ser desenvolvido seguirá o modelo *AP*, no qual o foco está na tolerância à partição e na disponibilidade, com eventual consistência do banco de dados. O banco suportará somente chaves e valores em formato de string. Além disso, as consultas serão simples, sem utilizar join por exemplo.

Justificativa

O grande volume de dados gerado por aplicações Web, juntamente com os requisitos diferenciados destas aplicações, como a escalabilidade sob demanda e o elevado grau de disponibilidade, têm contribuído para o surgimento de novos paradigmas e tecnologias [5]. Dentro deste cenário surgiu o modelo noSQL. As principais características deste modelo são: ausência de esquema ou esquema flexível, suporte nativo a replicação, API simples para acesso aos dados e consistência eventual [5].

Banco de dados noSQL fazem o tratamento de um conjunto limitado de funções, como pode ser observado na figura 1. Além disso, o modelo chave-valor é considerado de fácil implementação, permitindo que os dados sejam rapidamente acessados pela chave, principalmente em sistemas que possuem alta escalabilidade, contribuindo também para aumentar a disponibilidade de acesso aos dados [5].

O fato de cada sistema tratar diferentes problemas e ambientes, faz com que haja espaço para que outros sistemas especializados sejam criados e ainda assim, sejam relevantes.

Banco de dados comerciais que utilizam do método chave-valor

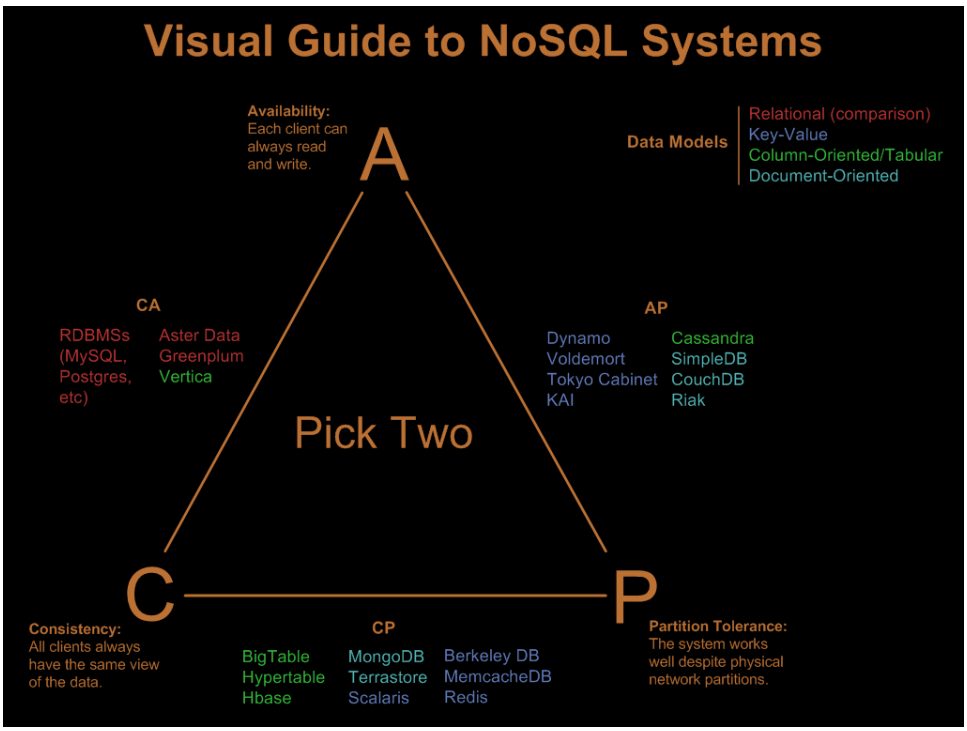

Figura 1: Guia visual do teorema CAP em sistemas NoSQL. Neste teorema, os sistemas tratam apenas dois dos três problemas. Em azul, estão representados os sistemas de chave-valor. Observe que diferentes sistemas possuem abordagens diferentes no que decidem como prioridade. Fonte [2].

restringem o tratamento de dados, por exemplo o Voldemort suporta json e string entre outras; DynamoDB suporta número, string, booleano, binário e conjuntos destes tipos; e o BerkeleyDB suporta um conjunto extenso de tipos de dados. O sistema proposto neste trabalho se diferencia por tratar apenas chaves e valores string, que aumenta a performance do sistema, eliminando testes e conversões desnecessárias, além de simplificar a implementação.

Para garantir a disponibilidade dos dados, são guardadas réplicas dos dados para dois nós afrente e dois atrás. Estas réplicas são atualizadas e removidas, conforme as operações realizadas sobre os dados. Assim, se o nó responsável por um determinado nó cair, e houver outro nó na rede, este irá assumir a responsabilidade pelos dados que antes eram cópias. De maneira semelhante, quando um novo nó entra na rede, ele ganha responsabilidade sobre uma faixa de dados, e se já houver dados na mesma, estes são enviados para o

nó.

Sendo assim, é possível garantir que o sistema garante a disponibilidade dos dados, caindo um nó por vez, dando um tempo para a rede se organizar e assumir os dados conforme necessário.

Referências

[1] Project-Voldemort - A distribuited database, http://www.project-voldemort.com/voldemort/

[2] Visual Guide to NoSQL Systems, http://blog.nahurst.com/visual-guide-to-nosql-systems

[3] Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis. BECKER, Georg. http://bit.ly/1J1mgB9

[4] Abordagem NoSQL – uma real alternativa. TOTH, Renato Molina http://bit.ly/1MhTflu

[5] NoSQL no desenvolvimento de aplicações Web colaborativas. LÓSCIO, B. F.; OLIVEIRA, H. R. de e PONTES, J. C. de S. http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf

[6] Using Merkle trees to detect inconsistencies in data http://distributeddatastore.blogspot.com.br/2013/07/cassandra-using-merkle-trees-to-detect.html

Tabela 1: Cronograma

Atividade	11-18/08	18-25	25-01	01-08	08-15	15-22	22-29	29-06	06-13	13-20	20-27	27-03	03-10	10-17	17-24/11
Estudar banco de dados noSQL	✓														
Estudar outros bancos chave-valor	✓														
Estudar técnicas de armazenamento distribuído	✓														
Definir a estrutura		✓													
Implementar instanciação do cluster		✓													
Distribuição			✓	✓	✓										
Replicação de dados					✓	✓									
Testes para replicação							✓								
Implementar inserção								✓							
Implementar busca									✓						
Tratamento de falhas										✓	✓				
Implementar atualização												✓	✓		
Implementar exclusão													✓		
Correção de bugs			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Tabela 2: Para cada etapa de implementação, será realizado baterias de testes para garantir a funcionalidade adequada do software. Repositório em https://github.com/Trindad/yek