Padrões e diretrizes arquiteturais para escalabilidade

Acrísio J. Nascimento Jr[1], Pedro Frosi Rosa[2], Ivens Oliveira Porto[2], Michel dos Santos Soares[2]

[1]Ciência da Computação – Universidade Federal de Goiás/Campus Catalão (UFG/CAC)
Caixa Postal 536 – 75.704-020 – Catalão – GO – Brazil.

[2]FACOM – Universidade Federal de Uberlândia (UFU) – Uberlândia, MG – Brazil.

acrisiojr@gmail.com, {frosi,michel}@facom.ufu.br,
ivens.porto@gmail.com

Abstract. *One issue not fully explored is how to build an architecture for a scalable system. There are works that discuss principles and general techniques for scalability, specially about performance improvement. However, this information is disorganized and unstructured, and are, to an architect with the responsibility of building a scalable system, a poor source of information to support his work.*

Resumo. *Uma questão não totalmente explorada é como construir uma arquitetura para um sistema escalável. Há obras que discutem princípios e técnicas gerais para a escalabilidade, especialmente sobre melhoria de desempenho. No entanto, esta informação é desorganizada e não estruturada, e são, para um arquiteto com a responsabilidade de construir um sistema escalável, uma fonte pobre de informação para apoiar o seu trabalho.*

1. Introdução

Nos dias atuais, escalabilidade tornou-se uma importante propriedade em sistemas. Com o volume crescente de usuários e dados que trafegam nas redes, os sistemas devem atender a esta demanda de maneira mais eficiente possível. Então, escalabilidade tornou-se um desafio para os arquitetos e desenvolvedores. Um exemplo de uso mais evidente em relação a esta propriedade são as ferramentas de busca (dentre elas, a mais popular: Google). Então, como arquitetar um sistema escalável?

O objetivo deste artigo é o de identificar, discutir as diretrizes e técnicas arquiteturais para auxiliar arquitetos de sistemas a projetar e construir sistemas escaláveis. As diretrizes e padrões deste trabalho são aplicáveis particularmente a sistemas web e sistemas de rede que trabalham com dados armazenados. Durante a fase de projeto, devem-se observar as diretrizes e técnicas de escalabilidade para que não se cometa erros no desenvolvimento. Não existe uma estratégia ou diretriz que solucione todos os problemas relacionados à escalabilidade.

Neste trabalho, as diretrizes e técnicas se preocupam principalmente com a escalabilidade horizontal, tornando possível a execução de um sistema em vários nós de processamento de tal maneira que, ao aumentar a quantidade de nós o sistema aumente, ou mantenha, seu desempenho de maneira satisfatória.

A escalabilidade horizontal tem um fator positivo , custo, que auxilia bastante no momento de determinar qual tipo de escalabilidade optar. Além da limitação a que um determinado nó se encontra para se escalar verticalmente.

A escalabilidade vertical possui algumas restrições, uma delas o custo, que inviabiliza seu uso se comparado com as demais formas de escalabilidade. Outro ponto importante é a complexidade de software em uma escalabilidade vertical.

2. Posicionamento de contexto em escalabilidade de sistemas

2.1. Categorias de Escalabilidade

Escalabilidade linear: significa que para cada recurso adicionado a um sistema, o desempenho aumenta de maneira diretamente proporcional, como mostra a Figura 1. O fator de escalabilidade aqui é igual a 1 [Williams, 2004].

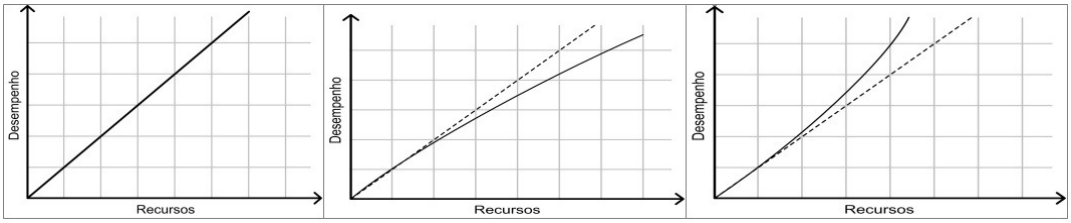

Figura 1. Escalabilidade linear, sublinear e superlinear

Escalabilidade sublinear: significa que para cada recurso adicionado a um sistema, o desempenho aumenta de maneira não proporcional, e inferior, à capacidade individual dos recursos adicionados, como mostra a Figura 1. O fator de escalabilidade é menor que 1 [Williams, 2004].

Escalabilidade super linear: como mostra a Figura 1 [Williams, 2004], significa que para cada recurso adicionado a um sistema, o desempenho aumenta de maneira superior à capacidade do recurso. O fator de escalabilidade é maior que 1. Isto se dá, porque quando se adiciona um determinado recurso, junto a este pode estar adicionado-se outros recursos.

2.3. Contexto de Escalabilidade

Construir sistemas de alto desempenho é diferente de se construir sistemas escaláveis. Para a construção de sistemas de alto desempenho aplicam-se técnicas para aumentar a velocidade de processamento e minimizar o uso de recursos pelo sistema, preocupando-se apenas em melhorar as métricas de desempenho de cada nó de processamento individual e não se considera a extensão do sistema como meio para aumentar o desempenho, como descrito na definição de escalabilidade horizontal.

No ano de 2000, Eric Brewer [Brewer, 2000] fez uma conjectura, que ficou conhecida como Teorema CAP, que diz o seguinte: “Dadas as propriedades de Consistência, Disponibilidade e Tolerância a Particionamento da rede, um serviço Web pode ter no máximo duas destas propriedades. Ela não se aplica apenas a serviços Web.

Consistência – um serviço é dito consistente se todos os clientes que acessam o objeto de dados vêem o mesmo dado, mesmo com a ocorrência de escritas concorrentes no objeto de dados.

Disponibilidade – para que um sistema distribuído esteja continuamente disponível, toda requisição recebida por um nó do sistema, que não apresente falhas, deve ter uma resposta. Em um contexto de acesso a dados, disponibilidade quer dizer que sempre será possível acessar o dado, mesmo que seja uma imagem (réplica) [Porto, 2009].

Tolerância a partições – as definições anteriores de Consistência e Disponibilidade são qualificadas com a necessidade de Tolerância a partições de rede [Porto, 2009].

A escolha das propriedades de Consistência e Disponibilidade resulta em sistemas onde é preciso que todos os nós devem comunicar-se para manter a consistência dos dados. Exemplos de sistemas deste tipo são: Bancos de Dados hospedados em um único local; e Bancos de Dados em *cluster* (ou qualquer outro tipo de arranjo). Algumas características marcantes destes sistemas são o uso de protocolo 2PC (*two phase commit*) e protocolos de invalidação de cache. Estes são sistemas distribuídos clássicos.

A escolha das propriedades de Consistência e Tolerância a Partições resulta em sistemas onde deve-se tolerar o fato de o sistema parar de responder quando ocorre um particionamento de rede, já que é preciso comunicação entre todos os nós para manter a consistência. Exemplos de sistemas deste tipo são Bancos de Dados distribuídos, Sistemas de Travas Distribuídas de dados. Algumas características marcantes destes sistemas são algoritmos de travas pessimistas, uso de protocolos de quorum, e o fato de que partições pequenas tornam o sistema indisponível.

A escolha das propriedades de Disponibilidade e Tolerância a Partições resulta em sistemas onde nem sempre se trabalha com dados atualizados. Exemplos de sistemas deste tipo são DNS (Domain Name System), caches WEB, Coda, Bayou [Demers et. al. 1994]. Suas características mais marcantes são uso de consistência temporal (TTL - time to live), uso de leases [Gray e Cheriton, 1989], algoritmos de travas otimistas e atualização otimista dos dados com possível resolução de conflitos.

3. Padrões Arquiteturais de Escalabilidade

Padrões de projeto (Design Pattern), em sua maioria, tratam de problemas relacionados a armazenamento de dados e uso de transações.

3.1. Definições e Aspectos

Elaborar arquiteturas de software não é uma tarefa fácil. Arquitetos e projetistas experientes quase nunca criam uma nova solução. Qualquer sistema é composto de subsistemas. Estes subsistemas disponibilizam uma interface para que possam ser utilizados por outros subsistemas. Quando subsistemas se tornam grandes e complexos, é interessante que se crie uma “fachada” de acesso ao subsistema, de tal modo que esta “fachada” unifique e facilite o uso do subsistema, como mostra a Figura 2 [Porto, 2009].

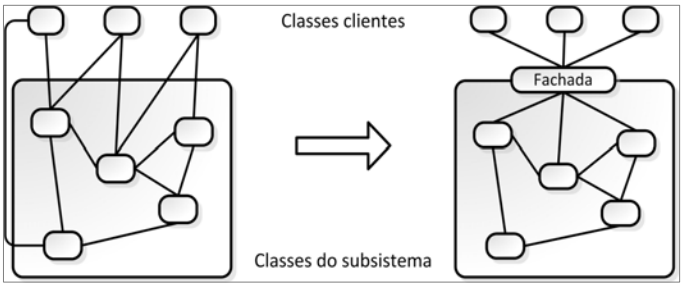

Figura 2. Padrão "fachada"

Uma definição ampla do que é um padrão, segundo [Porto, 2009], é a de que um padrão endereça um problema de projeto recorrente que surge em situações de projeto específicas, e apresenta uma solução. Padrão arquitetural: expressa o esquema organizacional estrutural fundamental de um sistema. Ele provê um conjunto predefinido de subsistemas, especifica suas responsabilidades, e inclui regras e diretrizes para organizar seus relacionamentos.

Um padrão de projeto é um esquema para refinar os componentes ou subsistemas de um sistema, ou o relacionamento entre eles. Ele descreve uma estrutura comumente recorrente de componentes que se comunicam para solucionar um problema genérico de projeto em um contexto particular [Porto, 2009], [Gamma et. al., 1995].

3.2. Vantagens no uso de padrões

Padrões documentam a experiência em projetos de software existentes, o que permite o uso sistemático da experiência adquirida em muitos anos de trabalho e auxilia arquitetos e projetistas menos experientes na criação de sistemas de melhor qualidade.

Identificam e especificam abstrações que estão acima do nível de classes individuais e instâncias de objetos e de componentes, e detalha suas responsabilidades e relacionamentos, sendo que estes componentes, juntos e em acordo como padrão, solucionam o problema colocado.

Estabelecem um vocabulário e um entendimento comuns dos princípios de projeto, onde os nomes dos padrões se tornam parte de uma linguagem de padrões que é compartilhada por todos e, esta linguagem facilita a comunicação, o entendimento do funcionamento do sistema, e a discussão de problemas de projeto, sendo que o nome do padrão pode abstrair sua complexidade e apenas com o nome ser possível comunicar toda a solução de um problema.

Auxiliam na construção de sistemas com propriedades definidas, além de tratar também de requisitos não funcionais, tais como escalabilidade, confiabilidade, reusabilidade, disponibilidade, etc.

3.3. Arquitetura Shared Nothing

Auxilia na construção de sistemas facilmente escaláveis na horizontal, a partir da estruturação do sistema em partes independentes, sem compartilhamento de estado (*Shared Nothing*), possibilitando a escalabilidade linear. Por exemplo, suponha um sítio onde o usuário forneça seus dados para cadastro e após isto, o mesmo seja autenticado na página para usufruir dos benefícios oferecidos pelo sítio. Normalmente, este tipo de

sistema é desenvolvido utilizando-se de uma arquitetura de 3 (três) camadas tradicional, sendo elas: apresentação, negócios e acesso a dados [Porto, 2009].

No intuito de aumentar o desempenho da solução, as camadas são fisicamente separadas, onde cada camada é composta por um *cluster* de servidores. As camadas de apresentação e de negócios têm seus dados relativos ao estado do uso do sistema pelo usuário armazenado em memória, como os que são utilizados por várias requisições.

Os servidores de cada camada se comunicam para sincronizar o trabalho e para aumentar a disponibilidade do sistema, além da comunicação entre as camadas para a sincronização de estado. À medida que uso do sistema cresce, adicionam-se novos servidores para atender à demanda, escalando-o horizontalmente.

A Figura 3 mostra o aumento da complexidade do sistema com a adição de novos servidores. Um modelo formal que descreve a escalabilidade linear é a lei de Amdahl [Amdahl, 1967].

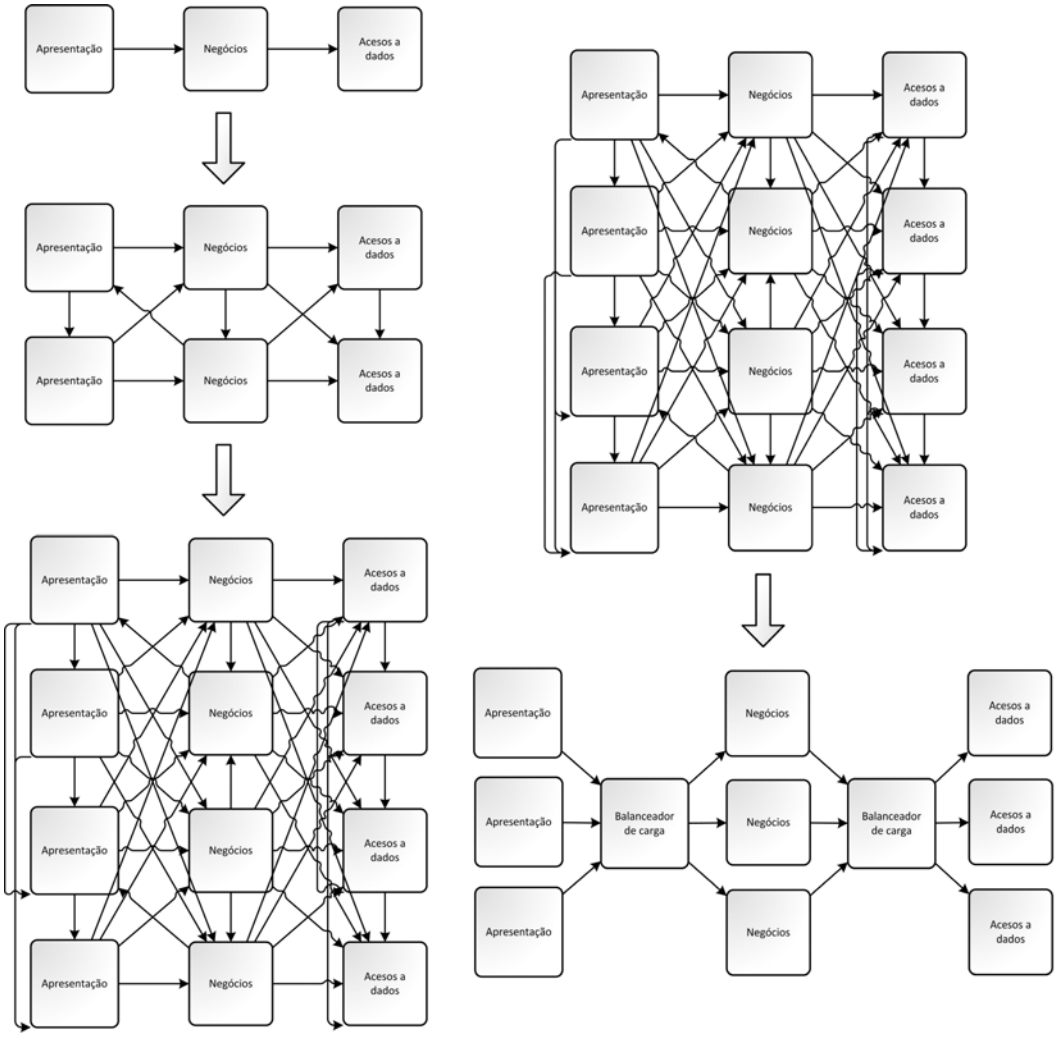

Figura 3. Aumento da complexidade do sistema com a adição de novos servidores e também uma possível solução com o uso de *Shared Nothing*

Podem haver sobretaxas de comunicação e sincronização, mas devem ser as menores possíveis. A idéia geral, em um sistema totalmente *Shared Nothing*, é remover toda a dependência entre os nós, como mostra a Figura 3.

A estrutura de uma Arquitetura *Shared Nothing* (SNA) é simples, basta que os nós do sistema sejam independentes. O resultado final da aplicação da SNA depende muito do problema, apesar do resultado ser sempre nós independentes, os nós resultantes e suas responsabilidades. A comunicação entre os nós de uma mesma camada não mais existirá, pois ter-se irá apenas comunicação entre camadas.

O padrão *Sharding*, mostrado na Figura 4, proporciona escalabilidade horizontal para os dados de um sistema, através do particionamento dos dados em vários bancos de dados, possibilitando uma alternativa de custo mais baixo e sem as limitações da escalabilidade vertical que normalmente é aplicada a banco de dados.

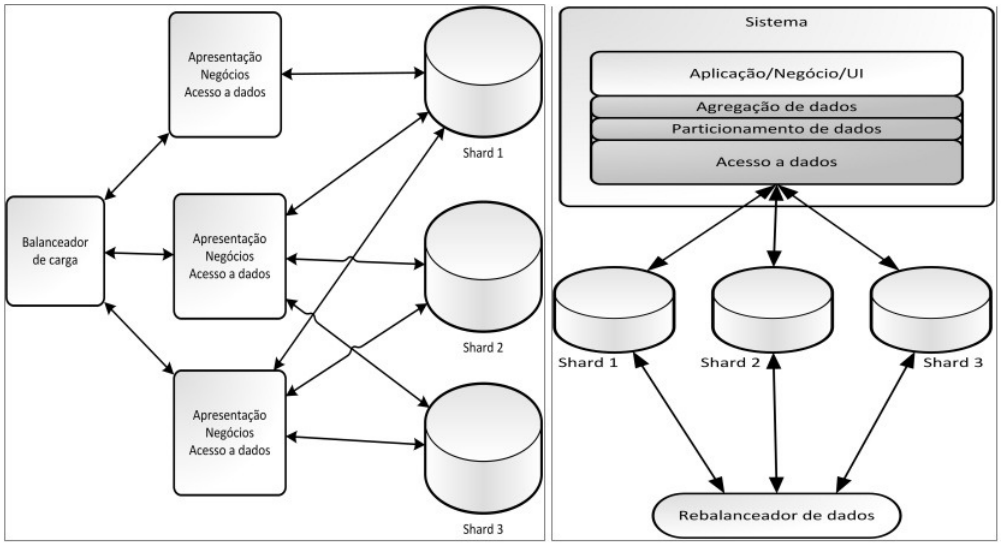

Figura 4. SNA com *Sharding* e a Arquitetura de um sistema com *Sharding*

Um fato relevante é que quando há a necessidade de escalar um banco de dados, ou a camada de dados de uma aplicação, o que geralmente se faz é adquirir computadores com maior capacidade de processamento, escalando verticalmente, e como se sabe esta abordagem é cara e limitada. Nesta situação, a opção é escalar o sistema horizontalmente, realizando uma distribuição dos dados.

Realizar *Sharding* de um banco de dados, como mostra a Figura 4, significa dividi-lo em instâncias menores, particionando e distribuindo os dados em um número de servidores de banco de dados. *Sharding* é um método de particionamento horizontal de dados que tem como objetivo a escalabilidade horizontal e o desempenho a um preço acessível. O rebalanceamento de dados nos *shards* pode ser implementado como um aplicativo separado, que monitora os *shards* e faz o rebalanceamento dos dados.

3.5. Arquitetura BASE

O padrão BASE (Figura 5) permite a construção de sistemas nos quais se troca a consistência de dados por escalabilidade ou disponibilidade, através da construção de um sistema que é basicamente disponível, lidando com dados ligeiramente

desatualizados e eventualmente consistentes. O uso de BASE em um sistema significa que o mesmo terá as seguintes propriedades:

- Basicamente disponível: o sistema terá tolerância a falhas parciais, mas não a falhas totais;
- Soft state: o sistema trabalhará com dados ligeiramente desatualizados;
- Consistência eventual: a propriedade de consistência, como especificada no Teorema CAP, não será respeitada, isto é, uma escrita em determinado dado não será visível a todo o sistema imediatamente, mas eventualmente o novo valor do dado será propagado para todo o sistema e todos os clientes e nós do sistema verão o mesmo dado [Vogels, 2008].

Talvez o uso mais conhecido do padrão BASE seja o sistema DNS (Domain Name System). Os dados são particionados entre vários servidores.

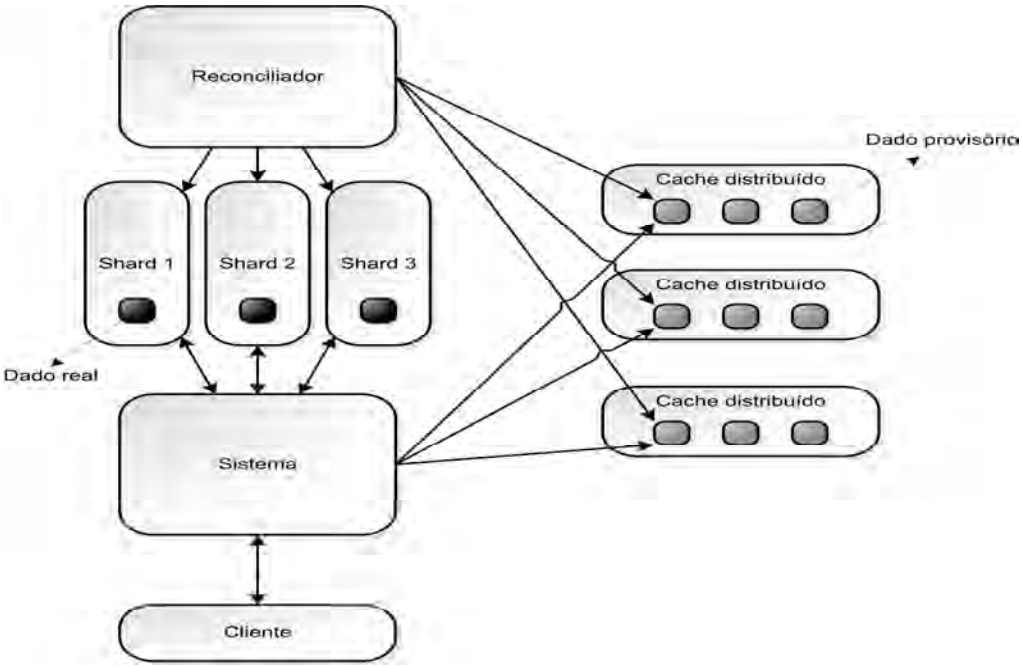

Figura 5. Estrutura de uma Arquitetura BASE com cache distribuído

4. Conclusão

Recentemente muita importância tem sido dada a sistemas escaláveis, pois praticamente todo sistema desenvolvido nos dias de hoje tem entre seus requisitos não funcionais a escalabilidade. O padrão Arquitetura *Shared Nothing* objetiva a construção de sistemas com escalabilidade linear através do uso de várias instâncias independentes do sistema evitando o aparecimento de gargalos. O padrão *Sharding* é uma alternativa para escalar os dados de um sistema, onde existem umas partes mais difíceis de serem escaladas horizontalmente. BASE é um padrão amplo que ataca vários problemas, onde se troca uma consistência forte dos dados por disponibilidade e tolerância a partições e se ganha em consequência escalabilidade e desempenho.

5. Referências

Amdahl, G. M., (1967). Validity of the single processor approach to achieving large scale computing capabilities. In AFIPS ’67 (Spring): Proc. of the April 18-20, spring joint computer conference, pp. 483–485, Atlantic City, New Jersey. ACM.

Brewer, E. A. (2000). Towards robust distributed systems (abstract). In PODC. ´00: Proc. of the nineteenth annual ACM symposium on Principles of distributed computing, p. 7, Portland, Oregon, United States. ACM.

Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., e Welch, B., (1994), The Bayou Architecture: Support for Data Sharing among Mobile Users.

Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). Design pat terns: elements of reusable object-oriented software. Addison-Wesley Professional.

Gray, C. e Cheriton, D., (1989). Leases: an effient fault-tolerant. mechanism for distributed file cache consistency. In SOSP ’89: Proceedings of the twelfth ACM symposium on Operating systems principles, pp. 202–210, ACM.

Porto, Ivens O. (2009), Dissertação de Mestrado sob o título “Padrões e diretrizes arquiteturais para escalabilidade de sistemas”, FACOM-UFU.

Vogels, W. (2008). Eventually Consistent. Queue, 6(6):14–19.

Williams, L. G. and C. U. S (2004). Web Application Scalability: A Model Based Approach. Technical report, Computer Measurement Group Conference (CMG’04).