@git
Advances in
Global Innovation & Technology

Arquitetura para Visualização e Interação com Mapas com Grandes Volumes de Pontos

Architecture for Visualization and Interaction with Maps with Large Point Volumes

Yehoshua Edson Oliveira Silva[1]
yehoshua.silva@fatec.sp.gov.br

Felipe Alves da Silva[1]
felipe.silva575@fatec.sp.gov.br

Leandro Luque[1]
leandro.luque@fatec.sp.gov.br

1 - Faculdade de Tecnologia de Mogi das Cruzes | Fatec Mogi das Cruzes

Recebido em
03 nov. 2023

Aceito em
15 fev. 2024

Publicado em
27 mar. 2024

https://git.fateczl.edu.br
e_ISSN
2965-3339
DOI
10.29327/2384439.2.2-11

@_GIT
Advances in Global
Innovation & Technology
Volume 2
Número 2
São Paulo
Março
2024

Resumo: O volume de dados gerados diariamente tem aumentado vertiginosamente nos últimos anos: estima-se que, em 2025, 400 exabytes serão gerados diariamente. Tal escala traz consigo problemas característicos – como desempenho e escalabilidade – e exige soluções apropriadas. Sendo parte desses dados espacialmente distribuídos usados amplamente em áreas como geoprocessamento, torna-se necessário também o cuidado com as técnicas de apresentação e usabilidade. Este artigo detalha uma arquitetura de software baseada em computação na nuvem, escalável e extensível para processamento e visualização de grandes volumes de dados geográficos. A solução proposta foi elaborada a partir da experiência conjunta dos autores em duas soluções de visualização geográfica, que são também discutidas de forma a propor melhorias e perspectivas para futuras implementações. Foram consideradas etapas primordiais da interação do usuário com dados espaciais: obtenção, agrupamento, exibição e registro de eventos. Levanto as principais questões arquiteturais, que originaram requisitos comuns funcionais e não funcionais a ambos os projetos.

Palavras-chave: Computação em nuvem; Arquitetura de software; Dados geográficos; Visualização de dados.

Abstract: *According to estimates, 400 exabytes of data will be generated daily in 2025. Such massive scale brings performance and scalability issues, requiring proper solutions. As part of this spatially distributed data is widely used in areas such as geoprocessing, it is also necessary to pay attention to presentation and usability techniques. This paper details a scalable, extensible, Cloud based software architecture for large scale geographical data processing and visualization. The proposed solution was developed based on the authors' joint experience in two geographic visualization solutions, which are also discussed in order to propose improvements and perspectives for future implementations. The primary stages of user interaction with spatial data were considered: obtaining, grouping, displaying and recording events. I raise the main architectural issues, which gave rise to common functional and non-functional requirements for both projects.*

Keywords: *Cloud computing; Software architecture; Geographical data; Data visualization.*

1. INTRODUÇÃO

O volume de dados gerados e processados por sistemas de *software* tem crescido exponencialmente nas últimas décadas. Taylor (2023) projeta que, em 2025, serão gerados mais de 400 exabytes – quintilhão de bytes – de dados diariamente. Muitos destes dados são espacialmente distribuídos (LEE e KANG, 2015) e, quando presentes em grande volume, impõem desafios não apenas quanto à experiência do usuário, mas também quanto à infraestrutura e aos métodos de armazenamento e processamento empregados.

Sistemas de *software* que gerenciam dados espacialmente distribuídos são vastamente empregados em campos como geoprocessamento, agricultura, logística e turismo, onde é crucial representar entidades em suas localizações específicas. Para aprimorar a experiência do usuário na visualização desses dados, várias técnicas podem ser adotadas. Algoritmos de clusterização, por exemplo, têm por função agrupar dados em grupos (ou *clusters*) de acordo com critérios preestabelecidos. Tal técnica possibilita a identificação e extração de informações até então desconhecidas (GOVENDER e SIVAKUMAR, 2019).

O processamento de dados em sistemas como estes torna-se viável tanto financeira quanto computacionalmente por meio da computação em nuvem - uma complexa infraestrutura de *software* e *hardware* que permite ao usuário acessar diversas plataformas, aplicações e serviços que estejam conectados a ela, permitindo a entrega da computação como serviço de forma transparente (PEDROSA E NOGUEIRA, 2011). Hashem et al. (2015) afirmam que a computação em nuvem oferece um conjunto único de soluções em armazenamento, processamento e análise de dados, fazendo com que arquiteturas baseadas em nuvem sejam cada vez mais adotadas em soluções acadêmicas e corporativas de grande escala.

Em 2021 e 2022, os autores do presente artigo contribuíram no desenvolvimento de dois sistemas que lidavam com consideráveis volumes de dados espaciais. Um estava relacionado ao transporte de cargas, representando mais de 130 mil veículos distribuídos por todo o Brasil. O outro era uma plataforma para uma grande empresa aérea brasileira, que auxiliava usuários a encontrar serviços diversos, sendo utilizado por centenas de milhares de usuários. Estes projetos permitiram validar diferentes propostas de arquitetura e estratégias de apresentação de dados espaciais.

Diversos trabalhos abordaram formas de lidar com grandes volumes de dados espacialmente distribuídos, porém, em sua grande maioria, focando em aspectos de apresentação de dados (DELORT, 2010; HUANG, 2012; KORPI, 2013; MEERT, 2006). A contribuição do presente artigo é apresentar uma arquitetura baseada em nuvem para processamento e visualização de dados espaciais oriunda de nossa experiência com os sistemas supracitados. Esta arquitetura tem como objetivo otimizar o tempo de resposta. Com este trabalho, esperamos proporcionar *insights* para profissionais da área, otimizando o processo de desenvolvimento ao reduzir a necessidade de explorar múltiplas alternativas não comprovadas.

2. FUNDAMENTAÇÃO TEÓRICA

Na interação de usuários com dados espaciais, algumas etapas precisam ser realizadas:

- Obtenção dos dados dos pontos (latitude, longitude, descrição etc.) a partir do nível de *zoom* de visualização;
- Opcionalmente, agrupamento destes pontos em *clusters* para redução do volume de informações visualizadas. No caso de agrupamento, obtenção dos dados dos grupos (latitude, longitude, tamanho e classificação);
- Exibição dos pontos ou *clusters* em uma região espacial, geralmente um mapa;
- Registro de eventos na região e nos pontos/*clusters* para responderem a interações do usuário, tais como: mudança de nível de *zoom*, seleção de ponto etc.

Algumas questões arquiteturais que influenciam estas etapas são:

- Onde e como armazenar os dados?
- De que forma os consultar?
- Em que momento e onde fazer a clusterização de pontos?
- Como responder eficientemente a movimentações feitas na região espacial, tanto em nível de translação quanto de *zoom*?

Orientada por estas questões, a arquitetura proposta neste artigo tem como pano de fundo a experiência dos autores em dois projetos profissionais, ambos com características e desafios próprios. Embora tivessem suas particularidades, os dois projetos contaram com alguns Requisitos Funcionais (RF) e Não-Funcionais (RNF) em comum:

- RF01: armazenar pontos de interesse com latitude e longitude e que podem ter categorias diferentes (p.ex.: hotel, aeroporto, locadora de veículos etc.);
- RF02: permitir aos usuários navegar por um mapa com os pontos, podendo visualizá-los e interagir com eles para obter mais informações;
- RF03: permitir que o usuário faça zoom no mapa, apresentando os pontos de modo a haver redução de sobreposição entre pontos;
- RNF01 (escalabilidade): o sistema deveria ser capaz de processar e apresentar volumes grandes e crescentes de dados, da ordem de centenas de milhares a milhões de pontos;
- RNF02 (desempenho): o sistema deveria apresentar os dados, independentemente de volume, dentro do intervalo limite de atenção de um usuário (NIELSEN, 2023).

Considerando estes requisitos comuns e as questões levantadas, estudamos diferentes estratégias, conforme tabela seguinte:

- Armazenamento dos pontos:
  - Banco de dados relacional;
  - Banco de dados *NoSQL*.
- Consulta de pontos:
  - Consulta por área de visualização (coletando apenas os pontos visíveis para o usuário no momento);
  - Consulta por área de visualização com margem (coletando pontos extras nas redondezas dos visíveis para o caso de movimentações serem respondidas mais rapidamente).
- Clusterização e armazenamento:
  - *Clusterizar* no *front-end*, distribuindo processamento entre as máquinas dos clientes;
  - *Clusterizar* no *back-end*, enviando aos usuários apenas as informações de clusters no nível de zoom atual do mapa em visualização pelo usuário.
- Resposta a interações dos usuários:
  - Montagem de clusters a cada interação;

- Manutenção de cache de clusters por nível de mapa;
- Uso de técnicas de *debouncing* para evitar requisições desnecessárias.

A programação e análise destas estratégias foi feita por meio de diversas iterações, gerenciadas por um processo SCRUM. As tecnologias utilizadas no *front-end e back-end* foram: *NodeJS*, *ExpressJS*, *ElasticSearch* e *Google Maps*.

Cada estratégia foi então analisada para um volume variável de 1.000, 10.000, 20.000, ..., até 100.000 pontos. A análise deu-se em função do tempo de resposta para o usuário, considerando as classificações propostas por Nielsen (2023). Por fim, definimos a arquitetura recomendada em função dos testes realizados como sendo aquela com o menor tempo de resposta.

Os autores devem seguir estritamente as normas descritas nesse *Template*. A norma a ser considerada é a NBR ABNT 6023 para citações e referências. O formato do autor-data deve ser usado à citação.

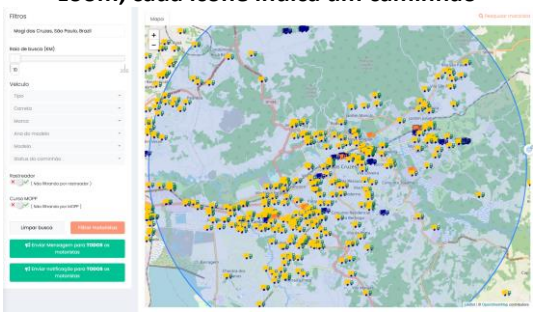
3. RESULTADOS E DISCUSSÃO

A primeira estratégia testada relacionava-se a um sistema monólito de logística, no qual era necessária a exibição e atualização em tempo real de milhares de pontos, representando a localização em tempo real de caminhões de transporte de carga, além de uma série de filtros de complexidade considerável. Nesta solução, utilizamos o banco de dados relacional PostgreSQL e os recursos do próprio Google Maps para exibição e clusterização automática de pontos. Ao navegar pelo mapa, mudando níveis de zoom, a API do Google Maps automaticamente atualiza os clusters, agrupando pontos próximos. Ao clicar sobre um cluster, o usuário era redirecionado para o nível de zoom mais próximo. Nos níveis bem próximos do mapa, cada caminhão era apresentado como um ponto distinto. Ainda, era possível filtrar caminhões por região, o que envolvia a exibição/remoção contínua de pontos do mapa. A escolha desta abordagem apresentou sérios problemas de desempenho com o crescimento do número de veículos (Figuras 1 e 2). Com 130 mil caminhões, o mapa demorava cerca de 20s para ser carregado e os pontos exibidos em um computador compatível com Intel I5 com 16GB de memória.

Figura 1 - Mapa com pontos *clusterizados*, indicando a quantidade clusters

Fonte: Os autores (2023)

Figura 2 - Detalhe do mapa em nível maior de zoom, cada ícone indica um caminhão



Fonte: Os autores (2023)

As dificuldades de desempenho enfrentadas com esta solução nos motivaram a uma refatoração de grande parte do código relevante. As mudanças efetuadas incluíram, mas não se limitaram a:

1. Implementação de um filtro geográfico baseado no *viewport* (janela de exibição, em tradução livre) mais atualizado do mapa: dividiu-se o mapa em três regiões concêntricas: a região 1 corresponde à parte visível do mapa. A região 2, a uma pequena margem de tamanho arbitrário. Caso o usuário navegue além da região 2, o conteúdo do mapa é recarregado conforme a nova região visível;
2. Refatoração da clusterização de pontos para o *back-end*. Em níveis baixos de zoom, grande parte do mapa é visível, aumentando a quantidade de pontos exibidos, caso o *front-end* seja o responsável pela clusterização. Tal refatoração permite reuso do processamento e reduz a quantidade de informações enviadas via internet;
3. Adição de otimizações relacionadas à interação do usuário com o mapa, das quais duas se destacam:
    a. Criação de cache dos clusters de acordo com o nível de zoom, de forma que pequenas mudanças não causem constantes requisições;
    b. Aplicação de *debouncing* nas chamadas à API, de forma que alterações rápidas de zoom ou localização no mapa não causem múltiplas requisições desnecessárias ao serviço.

A melhora de desempenho apresentada foi significativa, permitindo que a mesma experiência de navegação fosse alcançada, mas com limites superiores de tempo de 3s de abertura, na mesma configuração de máquina citada. Embora tenha apresentado desempenho maior, ainda existia o desafio da escalabilidade referente ao fato de que se tratava de um monólito.

O segundo projeto com o qual tivemos contato era uma solução de apoio na organização de viagens, com o objetivo de auxiliar o usuário final a encontrar hotéis, pontos de locação de veículos e atrações turísticas por todo o planeta Terra. As informações eram obtidas através das APIs de diversos parceiros, normalizadas em um formato padrão e disponibilizadas via API para uma página promocional.

A companhia aérea em questão possuía grande experiência com desenvolvimento Cloud e *serverless*, eliminando em grande parte os problemas de escalabilidade que uma solução desse gênero poderia causar.

Grande parcela da complexidade desse projeto estava localizada em alguns aspectos:

1. A integração com APIs parceiras, cada uma com seu contrato, detalhes de autenticação e ciclo de atualização dos dados;
2. Normalização das informações para consumo por parte do *front-end*;
3. A área de negócio da companhia desejava que os pontos de interesse fossem agrupados em diferentes níveis, em localizações específicas: clusters deveriam ser formados a nível de cidade, estados do Brasil, país e continente. O objetivo era facilitar a busca por parte do usuário final;
4. Os clusters supracitados deveriam ter localizações predeterminadas por parte da área de negócio.

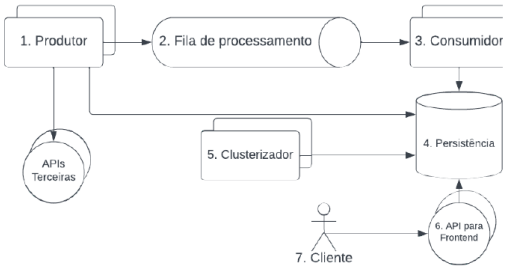
A arquitetura desenhada para esse projeto determinava a execução de um processo específico de consulta na API de cada um dos parceiros, todos agendados para iniciar durante a madrugada. Os dados eram então armazenados em um índice *ElasticSearch* específico, enquanto clusters eram representados como documentos no

mesmo índice. O armazenamento das localizações de cada cluster foi feito em um outro documento *ElasticSearch*.

Não houve necessidade posterior para refatoração, mas em retrospecto os autores indicam alguns pontos de melhoria:

- Implementação de uma etapa de configuração, removendo a configuração dos clusters do banco de dados;
- Processamento assíncrono dos dados, a fim de aumentar a responsividade e eficiência da solução.

Figura 3 - Nossa proposta arquitetural



Fonte: Os autores (2023)

Após a identificação desses requisitos comuns, definiu-se uma arquitetura base, facilmente alterada de acordo com a infraestrutura desejada (nuvem ou instalações próprias). A Figura 3 é uma visão geral dessa proposta, nela indicamos componentes escaláveis horizontalmente com a duplicação do ícone (componentes 1, 3. 5 e 6 são exemplos). O diagrama pode ser descrito em função de seus componentes essenciais:

1. Produtores: responsáveis por efetivamente produzir ou obter os dados base, que serão posteriormente inseridos em uma fila de processamento. Dados podem ser obtidos através de APIs externas, soluções IoT ou pré-processamento de arquivos proprietários, por exemplo;
2. Fila de processamento: itens produzidos são inseridos em uma fila para processamento assíncrono;
3. Consumidores: processam elementos retirados da fila, executando possíveis regras de negócio de acordo com o tipo do ponto, e executando a normalização, para depois persistir as informações;
4. Persistência: após uma etapa de normalização, os dados são armazenados em um banco de dados – preferencialmente otimizado para consultas;
5. *Clusterizador*: caso o algoritmo de clusterização escolhido não possa ser executado durante a etapa de processamento das informações, faz-se necessário um componente específico para tal. A frequência de execução deste componente definirá o ritmo de atualização das informações *clusterizadas*, a depender de requisitos não-funcionais e regras de negócio;
6. API para *front-end*: API para mediação entre o armazenamento escolhido e o *front-end* da aplicação. É efetivamente a implementação de um padrão arquitetural conhecido como BFF (ou *Backend-for-Frontend*), utilizado para abstrair lógica específica para a visualização. (BROWN e WOOLF, 2016);
7. Cliente: no *front-end* são implementadas otimizações especificamente relacionadas à interação do usuário com o mapa: *debounce* e cachês de requisição são exemplos de técnicas que podem ser utilizadas.

Em um ambiente Cloud, esta arquitetura pode ser facilmente escalada verticalmente, a partir do uso de recursos elásticos dos diferentes provedores de computação em nuvem. Exemplos são a *Amazon Web Services* (AWS), que fornece

um serviço de escalonamento automático de recursos provisionados chamado AWS *Auto Scaling* (AMAZON, 2018).

Por outro lado, a escalabilidade horizontal pode ser obtida através do uso de um orquestrador de contêineres, responsável por criar novas instâncias de acordo com o volume de requisições. Exemplos de orquestradores incluem *Docker Swarm* (DOCKER, 2023) e *Kubernetes* (KUBERNETES, 2023), sendo este último particularmente bem suportado por diversos provedores Cloud, que possuem serviços de provisionamento de clusters *Kubernetes*, como EKS (AMAZON, 2023), AKS (MICROSOFT, 2023) e GKE (GOOGLE, 2023).

Tipos diferentes de pontos podem ser tratados a partir da criação de Produtores específicos, com seus próprios ciclos de execução e forma de obtenção dos dados. Consumidores podem ser implementados a partir da mesma técnica, com a adição de configuração adicional na ferramenta escolhida para a fila. Cada tipo de ponto pode ter seu próprio ritmo de atualização dos dados, pois este depende apenas da frequência de execução dos Produtores e a etapa de processamento é assíncrona.

Uma etapa adicional de clusterização dos pontos pode ser executada juntamente às regras de negócio nos Consumidores. Xu e Wunsch (2005) listam e comparam uma série de algoritmos que podem ser usados para esse fim. Recomenda-se executar a clusterização em diversos níveis de granularidade, possibilitando a visualização de diferentes clusters de acordo com critérios arbitrários no front-end.

Uma etapa inicial de configuração pode ser criada para a customização do algoritmo utilizado, posição dos clusters e outros parâmetros. Os usuários da área de negócio de uma corporação podem então ser empoderados e responsabilizados por tal configuração (ALT et al., 2020).

4. CONCLUSÃO

Razavian et al. (2018) afirmam que a tomada de decisão é parte significativa da arquitetura de software – tão importante quanto a arquitetura em si é o processo de decisão que a originou, que pode e deve ser afetado pelas circunstâncias do projeto, pela área de negócio específica e pelo resultado desejado A arquitetura descrita neste artigo é derivada das experiências dos autores e resultante de suas observações e identificação de características comuns a soluções relacionadas ao processamento e visualização de dados geográficos: se trata de uma abordagem escalável e extensível, capaz de processar grandes volumes de dados e de fornecer uma base sólida para a experiência de usuário e de arquiteturas derivativas.

REFERÊNCIAS

ALT, Rainer; HUMAN, Soheil; NEUMANN, **Gustaf. End-user Empowerment in the Digital Age**. Proceedings of the 53rd Hawaii International Conference on System Sciences, 2020.

AMAZON. AWS **Application Auto Scaling.** 2023. Disponível em: https://aws.amazon.com/autoscaling/. Acesso em: 26 set. 2023.

AMAZON. **Managed Kubernetes Service - Amazon EKS Features**. 2023. Disponível em: https://aws.amazon.com/eks/features/. Acesso em: 26 set. 2023.

BROWN, Kyle et al. **Implementation Patterns for Microservices Architectures**. Conference on Pattern Languages of Programs, 2016.

DELORT, J. Visualizing large spatial datasets in interactive maps. In: 2010

SECOND INTERNATIONAL CONFERENCE ON ADVANCED GEOGRAPHIC INFORMATION SYSTEMS, APPLICATIONS, AND SERVICES, 2010, [S.l.]. Anais [...]. [S.l.]: [s.n.], 2010. p. 33-38. doi:10.1109/GEOProcessing.2010.13.

DOCKER. **Swarm mode overview**. 2023. Disponível em: https://docs.docker.com/engine/swarm/. Acesso em: 26 set. 2023.

GOOGLE. **Google Kubernetes Service (GKE).** 2023. Disponível em: https://cloud.google.com/kubernetes-ngine. Acesso em: 26 set. 2026

GOVENDER, Paulene; SIVAKUMAR, Venkataraman. **Application of k-means and hierarchical clustering techniques for analysis of air pollution: a review (1980-2019).** Atmospheric Pollution Research, 2020.

HASHEM, Ibrahim Abaker Targio et al. **The rise of "big data" on cloud computing: Review and open research issues**. Information Systems, 10 ago. 2014.

HUANG, H.; GARTNER, G. **A technical survey on decluttering of icons in online map-based mashups**. In: ONLINE MAPS WITH APIS AND WEBSERVICES. [S.l.]: Springer, 2012. p. 157-175. doi:10.1007/978-3-642-27485-5_11.

KORPI, J.; AHONEN-RAINIO, P. **Clutter reduction methods for point symbols in map mashups.** The Cartographic Journal, v. 50, n. 3, p. 257-265, 1 ago. 2013. doi:10.1179/1743277413Y.0000000065.

KUBERNETES. Overview. 2023. Disponível em: https://kubernetes.io/docs/concepts/overview/. Acesso em: 26 set. 2023.

LEE, J.-G.; KANG, M. **Geospatial big data: challenges and opportunities**. Big Data Research, [s.l.], v. 2, n. 2, p. 74-81, jun. 2015. DOI: 10.1016/j.bdr.2015.01.003.

MEERT, W.; TRONÇON, R.; JANSSENS, G. **Clustering maps**. 2006. Tese (Mestrado) - Katholieke Universiteit Leuven, Leuven, 2006. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.6977&rep=rep1&type=pdf Acesso em: 28 set 2023.

MICROSOFT. **Managed Kubernetes Service (AKS).** 2023. Disponível em: https://azure.microsoft.com/en-us/products/kubernetes-service. Acesso em: 26 set.2023.

NIELSEN NORMAN GROUP. **Website Response Times.** Disponível em: https://www.nngroup.com/articles/website-response-times. Acesso em: 28 set 2023.

PEDROSA, Paulo H. C.; NOGUEIRA, Tiago. **Computação em Nuvem**. Unicamp. 2011. Disponível em: https://www.ic.unicamp.br/~ducatte/mo40/1s2011/T2/Artigos/G04-095352-120531-t2.pdf. Acesso em: 26 set. 2023.

RAZAVIAN, Maryam; PAECH, Barbara; TANG, Antony. **Empirical Research for Software Architecture Decision Making, An Analysis**. The Journal of Systems & Software, 2019.

SVENNERBERG, G. **Handling large amounts of markers in google maps – in usability we trust**. 2009. Disponível em: http://www.svennerberg.com/2009/01/handling-large-amounts-of-markers-in-google-maps. Acesso em: 28 set. 2023.

TAYLOR, Petroc. **Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025**. Statista, 22 ago. 2023. Disponível em: https://www.statista.com/statistics/871513/worldwide-data-created. Acesso em: 28 set 2023.

XU, Rui; WUNSCH, Donald. **Survey of Clustering Algorithms**. IEEE Transactions on Neural Networks, 2005

SILVA, F. D. **Trabalhos científicos**. 2. ed. São Paulo: Genérica, v. 1, 2018.