Anatomia systemu rozproszonego

Od wymagań do architektury

mgr inż. Jakub Woźniak

Politechnika Poznańska · Instytut Informatyki · Semestr letni 2025/26

Plan wykładu

Agenda

- Co to znaczy *projektować* system rozproszony?
- Wymagania funkcjonalne vs niefunkcjonalne
- Back-of-the-envelope estimation
- Modele architektoniczne
- SLA / SLO / SLI
- Skalowanie: pionowe vs poziome

Projektowanie vs Operowanie

Rola architekta vs rola operatora

Architekt (PSR)

- **Jakie** komponenty?
- **Jak** się komunikują?
- **Gdzie** dane?
- **Dlaczego** ten trade-off?

Operator (ZSR)

- **Jak** wdrożyć na K8s?
- **Jak** monitorować?
- **Jak** zautomatyzować CI/CD?
- **Jak** reagować na awarie?

Czym jest system rozproszony?

Definicja

Zbiór niezależnych komputerów, które dla użytkownika wyglądają jak jeden spójny system.

Kluczowe konsekwencje:

- Brak współdzielonej pamięci
- Komunikacja przez sieć (zawodną!)
- Brak globalnego zegara
- Częściowe awarie (*partial failures*)

A. Tanenbaum, M. van Steen — Distributed Systems, *3rd ed.*

Wymagania

Wymagania funkcjonalne vs niefunkcjonalne

Funkcjonalne

Co system ma robić?

- Złożenie zamówienia
- Wysłanie potwierdzenia
- Historia transakcji

Niefunkcjonalne

Jak dobrze ma to robić?

- **Reliability** — mimo awarii
- **Scalability** — mimo wzrostu
- **Maintainability** — łatwość zmian

Trzy filary (Kleppmann)

Designing Data-Intensive Applications, rozdz. 1

1. **Reliability** — system działa poprawnie nawet gdy coś idzie nie tak
2. **Scalability** — radzi sobie ze wzrostem obciążenia
3. **Maintainability** — ludzie mogą efektywnie z nim pracować

M. Kleppmann — Designing Data-Intensive Applications, *O'Reilly, 2017*

Back-of-the-Envelope Estimation

Szacowanie „na serwetce“

Zanim projektujesz — **policz**.

Pytania na start

- Ilu mamy użytkowników? (DAU)
- Ile żądań/s? (QPS)
- Ile danych dziennie / rocznie?
- Stosunek odczytów do zapisów?
- Akceptowalne opóźnienie? (p50, p99)

Rzędy wielkości — opóźnienia

Operacja	Opóźnienie
Odczyt z RAM	100 ns
Odczyt z SSD	100 µs
Odczyt z dysku (HDD)	10 ms
Round-trip w DC	0.5 ms
Round-trip transatlantycki	150 ms

Rzędy wielkości — przepustowość

System	QPS
1 serwer webowy	1K–10K
1 instancja PostgreSQL (odczyty)	5K–20K
1 instancja Redis	100K
Kafka (1 partycja)	100K msg/s

Przykład: system powiadomień push

Dane wejściowe

10 mln użytkowników, średnio 5 powiadomień/dzień.

Szacunki:

- Powiadomień/dzień: 10M × 5 = **50M**
- Średnio na sekundę: 50M / 86 400 ≈ **580/s**
- Szczyt (10× średniej): **5 800/s**
- Storage (100B × 30 dni): **150 GB**

Estimation w chmurze — ile to kosztuje?

Cloud pricing = nowy wymiar estimation

Oprócz QPS i storage — w chmurze liczymy **koszt**.

Zasób	Cena (orient.)	Nasz system/mies.
Lambda (1M req)	$0.20	50M × 30 ≈ $300
S3 storage (GB)	$0.023	150 GB ≈ $3.50
SQS (1M msg)	$0.40	1.5B ≈ $600

Modele architektoniczne

Client-Server i Peer-to-Peer

Client-Server

Serwer odpowiada na żądania.

- Prostota, centralna kontrola
- Single point of failure
- Np. REST API + SPA

Peer-to-Peer

Każdy węzeł = klient i serwer.

- Brak centralnej awarii
- Trudna spójność danych
- Np. BitTorrent, blockchain

Event-Driven i Hybrid

Event-Driven

Komunikacja przez zdarzenia.

- Luźne powiązanie
- Asynchroniczność
- Np. Kafka + mikroserwisy

Hybrid

Łączenie modeli.

- REST — API publiczne
- Eventy — wewnętrznie
- P2P — CDN/edge

SLA / SLO / SLI

Kontrakt niezawodności

Definicje

- **SLI** — metryka: np. opóźnienie p99, error rate
- **SLO** — cel: p99 opóźnienia < 200ms w 99.9% czasu
- **SLA** — kontrakt z konsekwencjami (SLO + kary)

Google SRE Book — Service Level Objectives, *rozdz. 4*

„Dziewiątki“ dostępności

Dostępność	Downtime/rok	Downtime/mies.
99% (dwie 9)	3.65 dni	7.31 h
99.9% (trzy 9)	8.77 h	43.83 min
99.99% (cztery 9)	52.60 min	4.38 min
99.999% (pięć 9)	5.26 min	26.30 s

Skalowanie

Pionowe vs Poziome

Scale Up (pionowe)

Większy serwer.

- ☒ Brak zmian w kodzie
- ☒ Fizyczne limity
- ☒ Single point of failure

Scale Out (poziome)

Więcej serwerów.

- ☒ Nieograniczone (teoria)
- ☒ Redundancja
- ☒ Wymaga stateless design

Zasada nr 1 skalowania poziomego

Serwisy powinny być **bezstanowe** (stateless) — cały stan w zewnętrznym systemie (baza, cache, object storage).

Gdy stan jest nieunikniony:

- **Sticky sessions** — ten sam user ⟶ ta sama instancja
- **Zewnętrzny session store** — np. Redis
- **Consistent hashing** — deterministyczne przypisanie

Skalowanie w chmurze — managed services

Nie skaluj sam — daj chmurze skalować za Ciebie

Managed services przejmują operacyjną złożoność skalowania.

Self-managed:

- Ręczne repliki DB
- Własny Redis cluster
- Konfiguracja LB + health checks
- Budzenie się o 3 w nocy

Cloud-managed:

- RDS Multi-AZ + Read Replicas
- ElastiCache / Cloud Memorystore
- ALB / Cloud Load Balancing
- Autoscaling + alerting

Przykład: Scale From Zero To Millions

Ewolucja architektury (A. Xu)

System Design Interview, rozdz. 1

Etapy wzrostu typowej aplikacji webowej:

1. **1 serwer** — web + DB na jednej maszynie
2. **Separacja DB** — dedykowany serwer bazodanowy
3. **Load balancer** — 2+ serwery web, repliki DB
4. **Cache** — Redis/Memcached dla gorących danych

Ewolucja — kolejne etapy

1. **CDN** — statyczne zasoby bliżej użytkownika
2. **Kolejka wiadomości** — asynchroniczne zadania
3. **Sharding bazy** — podział danych na partycje

Kiedy co wprowadzać?

Sygnał	Rozwiązanie
CPU DB > 80%	Repliki odczytu
Opóźnienie rośnie z ruchem	Load balancer + instancje
Te same dane czytane 100×	Redis / Memcached
Użytkownicy w innych regionach	CDN + multi-region
Timeouty przy wysyłce mail/SMS	Kolejka wiadomości

Projektujesz system powiadomień push dla aplikacji z 10 mln użytkowników.

Jakie metryki oszacujesz na samym początku?

Jakie SLO zaproponujesz — i *komu*?

Podsumowanie

Kluczowe wnioski

1. **Projektowanie ≠ operowanie** — decyzje *przed* wdrożeniem
2. **Wymagania niefunkcjonalne** kształtują architekturę
3. **Policz zanim projektujesz** — estimation na serwetce
4. **SLO definiuj od dnia pierwszego**
5. **Scale out > scale up** — wymaga stateless design

Źródła i lektury

- M. Kleppmann — *DDIA*, rozdz. 1
- A. Xu — *System Design Interview*, rozdz. 1–3
- Google SRE Book, rozdz. 4
- M. Takada — *Distributed Systems for Fun and Profit*, rozdz. 1

Następny wykład

Wykład 2: Style architektoniczne

Monolit · Mikroserwisy · Modularny monolit