Dokumentace IMS-2024/25

Simulace Škálování webové služby - Reaktivní vs Prediktivní přístup

Datum:
27.11.2024

Autoři:
Jakub Fukala - xfukal01
Adam Kozubek - xkozub09

1. Úvod

Smyslem této práce je vytvořit simulaci (slide č.33) škálování webové služby při proměnných podmínkách. Simulace bude porovnávat dva přístupy škálování (reaktivní a prediktivní). Na základě výsledků simulace vzorových modelů jsme stanovili obecné poznatky a demonstrovali za jakých okolností je lepší využít který přístup. Vytvořený simulační nástroj je ale určen primárně pro firmy na pomoc při rozhodování, kterou podobu škálování a s jakým nastavením je vhodné použít pro konkrétní službu.

Bylo nutné okrajově nastudovat fungování algoritmů Autoscalingu, zjistit reálné režie jednotlivých procesů a především vytvořit co nejpřesnější model (slide č.44) předpokládané vytíženosti a na základě studia dalších podkladů, odhadnout jeho odchylku oproti reálné vytíženosti.

1.1 Autoři a významné zdroje

Autory práce jsou Jakub Fukala a Adam Kozubek, kteří ji vytvořili jako projekt do předmětu modelování a simulace na FIT VUT v Brně. Významnými zdroji práce jsou zejména **výukové zdroje předmětu** IMS [8] [9].

Jako reprezentativní dataset vytížení byl pro naši práci použit reálný záznam denního vytížení z **Internet traffic archivu** [4]. Ten jsme dále zpracovali pomocí python knihoven.

Pro vytvoření relevantních modelů ??? autoscaleru, loadbalanceru a správné implementace obou přístupů škálování, jsme studovali internetové zdroje a knižní zdroj **The Datacenter as a Computer** [14]. Všechny použité zdroje jsou uvedeny na konci této dokumentace v sekci Bibliografie.

1.2 Prostředí a validace modelu

Model je vytvořen v jazyce **C++** a používá knihovnu **SIMLIB/C++** [8]. Ačkoliv je samotná simulace nedeterministická, její běh dá na různých strojích se stejnými vstupy srovnatelné výsledky, resp. nalezená odchylka je totožná s odchylkou (slide č.82) mezi více výstupy téže zopakované simulace na témže stroji.

Ačkoliv je model zjednodušený, má 15 nastavitelných parametrů a odpovídá realitě, protože byl vytvořen a nastaven na základě důkladné analýzy dostupných zdrojů. Také výstup simulace za daných podmínek byl srovnán s reálnými poznatky. Na základě toho model považujeme za validní (slide č.36).

2. Rozbor tématu a použitých metod a technologií

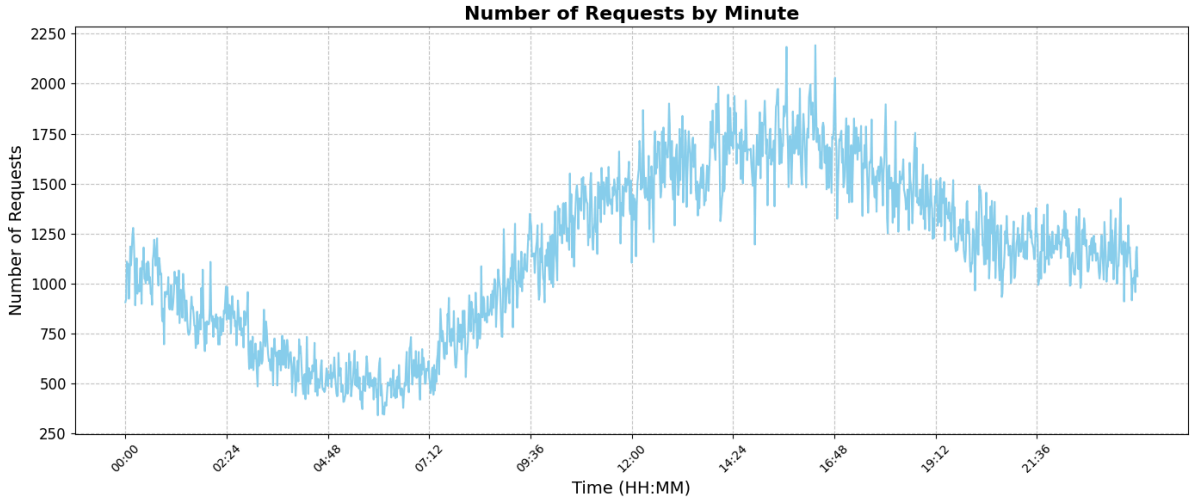
Moderní aplikace jsou stále častěji budovány na principu **mikroservisní architektury** [10], která umožňuje rozdělení aplikace na nezávislé komponenty – **mikroslužby**. Tato architektura umožňuje nejen nezávislý vývoj a nasazování jednotlivých částí systému, ale také jejich nezávislé škálování, čímž je dosaženo efektivního využití zdrojů. Klíčovou výzvou v prostředí mikroslužeb je **efektivní škálování**, které zajišťuje splnění dohodnuté úrovně služeb (**SLA**) [5] a zároveň minimalizaci nákladů na provoz infrastruktury.

Pro náš model jsme hledali relevantní hodnoty parametrů pro klasickou webovou službu, na základě studií uvedených zdrojů jsme zvolili následující nastavení modelu:

- Typické webové aplikace mohou mít průměrné zatížení mezi **20 % až 60 %** CPU během běžného provozu, s krátkodobými špičkami až 90 % CPU při neočekávaných událostech [10].
- Vytvoření a zpřístupnění nového kontejneru na běžné platformě, jako je Kubernetes, trvá mezi **5 až 15 sekundami** v ideálním případě. Při zatížení systému se tento čas může zvýšit až na 30 sekund [10].
- Latence přesměrování: Dynamické algoritmy LoadBalanceru, jako Weighted Round Robin, mají přidanou latenci přesměrování obvykle **<10 ms** na požadavek. Tato hodnota se může zvýšit při přetížení na 30-50 ms [6].
- Směrodatná odchylka u odhadovaného zatížení (např. pro metriky CPU) se typicky pohybuje mezi **5 % až 25 %**, v závislosti na kvalitě dat a složitosti modelu [6].
- Reaktivní autoscalery reagují obvykle v časech **1-5 minut**, což zahrnuje čas na detekci potřeby škálování a spuštění nových instancí. [1]
- SLA jsme zvolili jako **95%** požadavků zpracovaných do **200 ms** [5].

Všechny tyto parametry společně s dalšími (podmínkami pro přeškálování, …) jsou nastavitelné v sekci –PARAMETRY SIMULACE– tak, aby šla celá simulace spustit s modelem relevantním jiné konkrétní službě.

Zdrojem dat zátěže byl **Internet traffic archiv** [4], který nám poskytl reálná historická data počtu požadavků naměřených na konkrétním servery. Naměřený vzorek dat obsahoval záznam za sedm dní. Jednotlivé požadavky byly pro potřeby simulace shlukovány po jedné minutě a následně byla vytvořena průměrná hodnota požadavku v dané minutě. Výsledek byl vynesen do grafu o časové délce jednoho dne.
Zde je grafické znázornění vytvořené datové sady:



Historická data zátěže jsou nahrána do prediktivního modelu.

2.1. Popis použitých postupů pro vytvoření modelu a zdůvodnění

Simulace využívá modelování **škálování aplikací v kontejnerizačním prostředí**, kde je využito jak reaktivního, tak prediktivního přístupu k řízení počtu instancí. Reaktivní škálování reaguje na aktuální celkovou hodnotu zatížení kontejneru, zatímco prediktivní škálování využívá **historická data** a metody k předpovědi budoucí zátěže.

Postup modelování:

- **Generování požadavků:** Je dáno exponenciálním rozdělením (slide č.91) [2] [9]. Byly použity reálné datové soubory pro distribuci požadavků během 24 hodin [4].
- **Horizontální škálování:** Model škáluje počet instancí (kontejnerů) na základě průměrné zátěže (tj. průměrný počet současně odbavovaných požadavků). Doba spuštění nového kontejneru odpovídá hodnotám měřeným v prostředích Docker a Kubernetes [2].
- **Latence:** V modelu je zohledněno, že latence při obsluze požadavků narůstá při zvyšování zátěže daného kontejneru [14]. Pro zjednodušení jsme použili nejjednodušší model lineárního nárůstu [12].
- **Porovnání přístupů:** Simulace zahrnuje srovnání reaktivního a prediktivního přístupu. Prediktivní škálování dokáže díky předpovědím snížit riziko přetížení při náhlých nárůstem zátěže, zatímco reaktivní přístup může být efektivnější při minimalizaci nákladů ve stabilním prostředí [14].

Zdůvodnění postupů: Reaktivní škálování je jednodušší na implementaci a dobře vyhovuje scénářům s předvídatelnou a stabilní zátěží. Naopak prediktivní škálování vyžaduje více výpočetních zdrojů a kvalitní historická data, ale dokáže předcházet přetížení ve špičkách. Oba přístupy jsou v modelu implementovány a simulovány, aby byly vyhodnoceny jejich výhody a nevýhody v různých scénářích.

2.2. Popis původu použitých metod/technologií

Použité metody a technologie byly vybrány na základě ověřených vědeckých zdrojů a zkušeností z provozních systémů:

- **Kontejnerizace:** Simulace je navržena s ohledem na vlastnosti moderních kontejnerizačních nástrojů, jako je Docker a Kubernetes [2].
- **Škálování:** Pravidla škálování (např. prahové hodnoty) byla navržena na základě běžných praktik [1] a vlastních výsledků simulace.
- **Generování zátěže:** Datová sada obsahující záznamy o počtu požadavků za minutu pochází z reálných provozních měření a byla přizpůsobena pro simulaci pomocí normálního rozptylu s 15% odchylkou [4] [6].
- **LoadBlanacing:** Jsme implementovali jednoduše tak, že nový požadavek je přiřazen kontejneru s nejnižším zatížením.

Tento přístup zajišťuje validitu (slide č.37) modelu a poskytuje užitečné srovnání škálovacích metod, které mohou být aplikovány v reálných systémech. Na základě těchto simulací lze posoudit efektivitu různých strategií škálování a jejich dopad na náklady i kvalitu služeb.

Pokročilé matematické funkce a funkce obsluhující běh simulace byly použity z knihovny **SIMLIB/C++** (slide č.38) [8], která je přesně na tento účel vytvořená.

3. Koncepce - modelářská témata

Náš model zanedbává dva aspekty. Prvním je náběh systému, kdy je připravený defaultně jediný kontejner a škálovací modely musí rychle zvýšit jejich počet, neboť zátěž je normální. Druhým je fakt, že latence od určitého bodu zátěže nenarůstá lineárně, respektive tento koncept nelze uvažovat, když chceme modelovat reálnou latenci.

Důvodem tohoto zjednodušení je fakt, že validita modelu není ovlivněna. Model ale není určen pro simulaci extrémních stavů, či náběhu a ukončování systému. Při náběhu systému se rychle dosáhne rovnováhy a v celkových statistikách se tento efekt neprojeví. Totéž platí o ukončování systému.

Dále se model omezuje jen na homogenní kontejnery, to znamená, že všechny kontejnery jsou považovány za identické z hlediska výkonu a latence, což zjednodušuje výpočty, ale odpovídá praxi v cloudovém prostředí s předdefinovanými instancemi.

Také jsou ignorovány síťové latence. Síťová latence je považována za zanedbatelnou, neboť většina aplikací má nízkou komunikaci mezi kontejnery v jednom uzlu.

Pro naše účely vyvození závěrů při běžném provozu systému běžné webové služby tak je model validní a dané závěry relevantní.

3.1 Způsob vyjádření konceptuálního modelu

Konceptuální model simuluje dynamické škálování aplikací v prostředí mikroservisní architektury. Tento model abstrahuje klíčové vlastnosti systému, jako je zátěž, odezva aplikací, přidávání a odebírání kontejnerů a jejich čas spuštění, a redukuje je na měřitelné a simulovatelné metriky. Model je vyjádřen kombinací schématického znázornění, stavového diagramu, Petriho sítí a matematických rovnic:

- **Schéma systému**: Na *obrázku 1* je znázorněn obecný tok požadavků od uživatele až po zpracování v rámci kontejnerů. Schéma ilustruje, jak zátěž vstupuje do systému, je rozdělena mezi aktivní kontejnery a třemi tečkami je naznačeno, jak probíhá škálování.

- **Matematické rovnice**:
  - Lineární nárůst latence zpracování požadavku se zátěží kontejneru

    $$doba\_zpracování = min\_doba\_obsluhy \cdot (1 + \alpha \cdot zátěž\_kontejneru)$$

  - Odchylka reálných dat zátěže oproti predikci

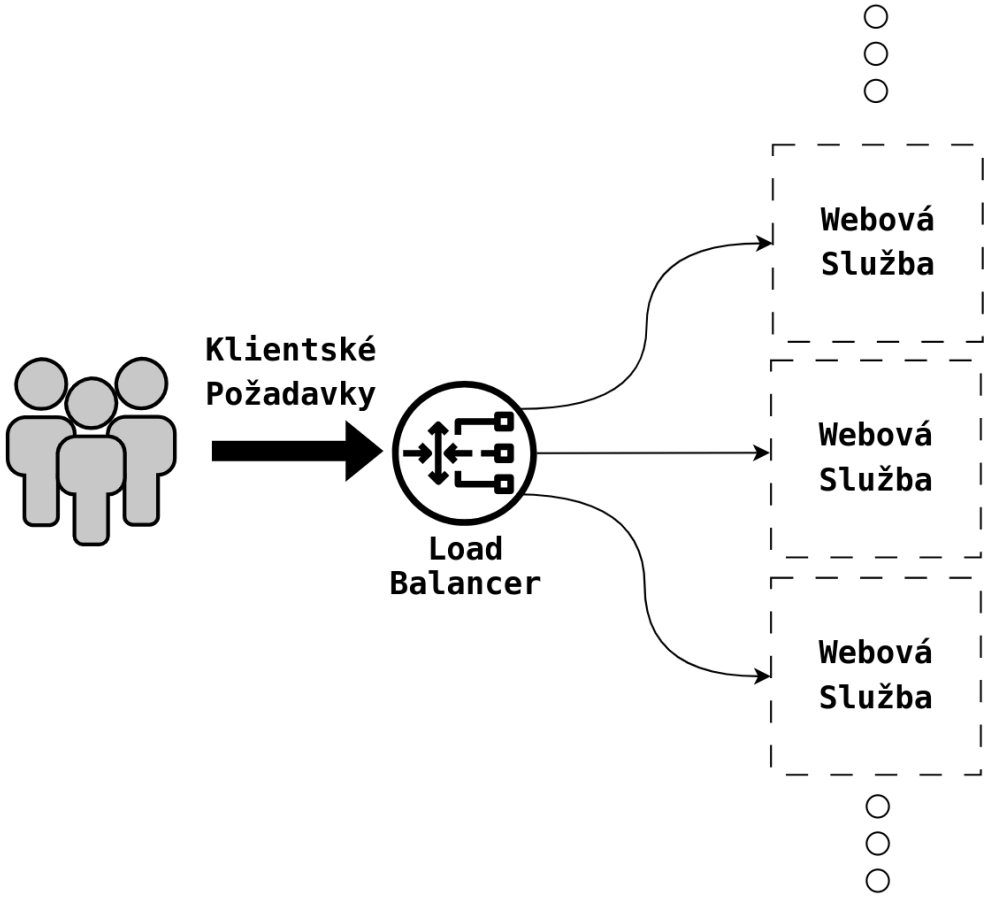
    $$real_{LOAD} = Norm\,(predikce_{LOAD};\ \sigma \cdot predikce_{LOAD})$$

- **Stavový diagram**: Na *obrázku 2* je znázorněn stavový diagram pro jednotlivé kontejnery, které mohou být v jedné z následujících stavů: **Neaktivní**, **Spouštění**, **Připravený** nebo **Aktivní** Diagram také zahrnuje přechody mezi stavy, například

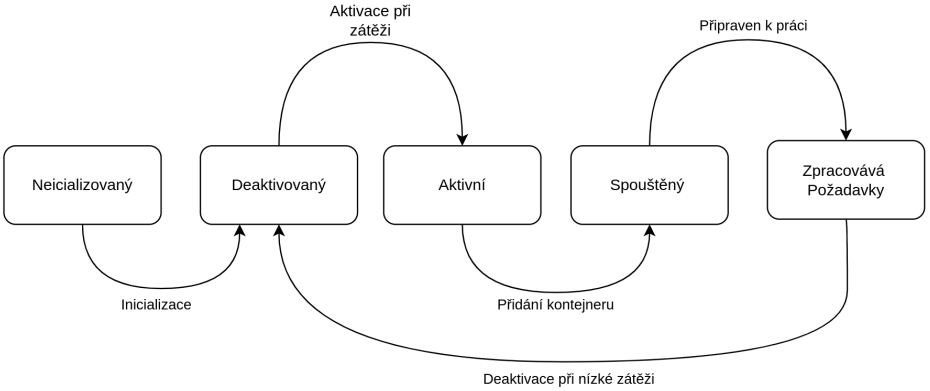
aktivaci při překročení zátěže nebo deaktivaci při nízkém vytížení.

- **Petriho síť:** Na *obrázcích 3* a *4* je znázorněna abstraktní Petriho síť modelu. Konkrétně na *obrázku 3* je část s LoadBalancerem a na *obrázku 4* je část s Autoscalerem. Tyto dvě části jsou propojené skrze místo Aktivní kontejnery, které je spravované AutoScalerem. Požadavky jsou generované s exponenciálním rozdělením, jehož středem je funkce *f(t)*, která odpovídá datasetu zatížení. Modely pro škálování zde jsou uvedeny pouze abstraktně v Autoscaleru v podmínkách přechodů pro přidání/odebrání kontejneru.

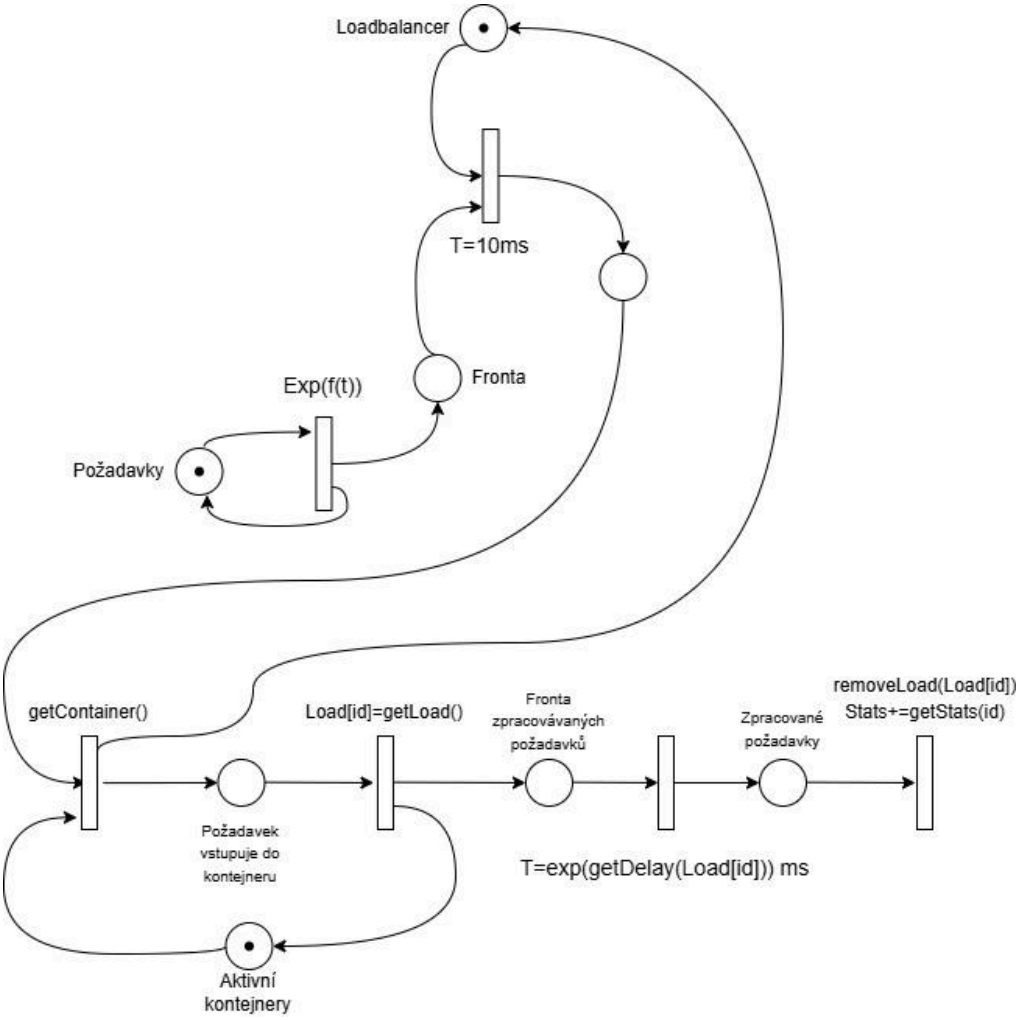
3.2 Forma vyjádření konceptuálního modelu



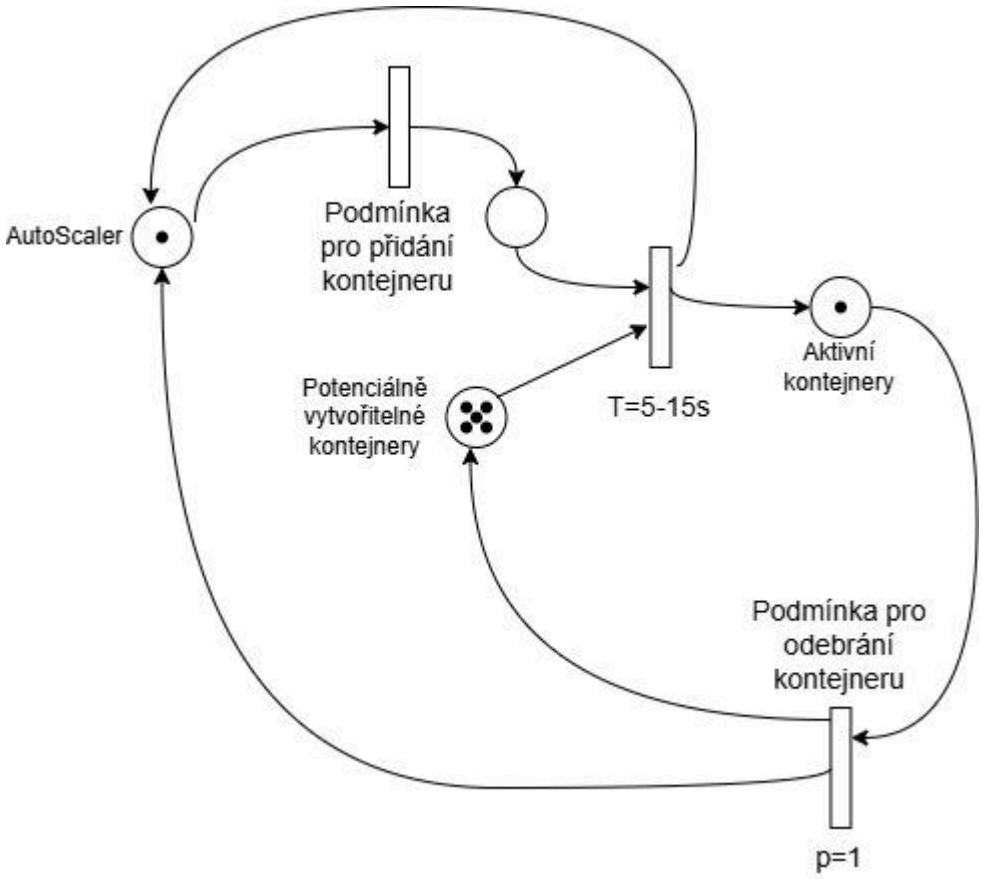
(Obrázek: 1 - Schéma systému)



(Obrázek: 2 - Stavový diagram kontejneru)



(Obrázek: 3 - Petriho síť (slide č.124) *- LoadBalancer)*



(Obrázek: 4 - Petriho síť (slide č.124) *- AutoScaler)*

3. Koncepce - implementační témata

IDK 😣

4. Architektura simulačního modelu/simulátoru

4.1 Mapování konceptuálního modelu do simulačního modelu

V této kapitole je podrobně popsáno, jak byl konceptuální model z kapitoly 3 implementován do konkrétního simulačního modelu za použití knihovny Simlib. Implementace využívá objektově orientované programování k modelování klíčových procesů a veličin. Níže jsou popsány jednotlivé prvky konceptuálního modelu a jejich odpovídající implementace v simulačním modelu.

Generátor požadavků

- **Konceptuální model**: Proces generuje příchozí požadavky podle dané zátěže modelované jako normální rozdělení na základě historických dat.
- **Implementace**:
    - **Třída `RequestGenerator`** dědí od `Event`. Aktivuje procesy požadavků v časových intervalech definovaných metodou `GetInterarrivalTime`, která využívá reálná data z pole `real_requests_per_minute`.
    - **Reálná data**: Data o zátěži jsou generována funkcí `GenerateRealRequestsPerMinute` a zohledňují variabilitu s odchylkou 15 %.

Požadavek

- **Konceptuální model**: Požadavek čeká na přidělení volného kontejneru a je následně zpracován. Doba zpracování závisí na aktuální zátěži přiděleného kontejneru.
- **Implementace**:
    - **Třída `Request`** dědí od `Process`. Obsahuje metody pro simulaci přiřazení požadavku a zpracování v kontejneru.
    - **Výpočet doby zpracování**: Doba zpracování je modelována vztahem:

      $$t_{response} = t_{service} \cdot (1 + \alpha \cdot Load)$$

      kde α\alphaα je koeficient ovlivňující nárůst latence při vyšší zátěži.

Kontejner

- **Konceptuální model**: Kontejner má stavy **neaktivní**, **spouštějící**, **připravený** a **zpracovávající**. Sleduje zátěž, celkovou dobu aktivity a stav připravenosti.
- **Implementace**:
    - **Třída `Container`** reprezentuje jednotlivé kontejnery. Obsahuje atributy jako `is_ready`, `load` a `total_active_time`.
    - Metody:
        - `Activate` a `Deactivate`: Mění stav kontejneru.

        - `AcceptRequest` a `ReleaseRequest`: Přijímání a uvolnění požadavků.
        - `Start`: Nastavuje stav připravenosti po dokončení spouštění.
    - **Spouštění kontejneru**: Proces spouštění je reprezentován třídou `ContainerStartup`, která dědí od `Event`.

Škálování

- **Konceptuální model**: Systém obsahuje dva různé škálovací algoritmy:
    - **Reaktivní škálování** reaguje na aktuální zátěž.
    - **Prediktivní škálování** využívá historická data k predikci budoucí zátěže.

- **Implementace**:
    - **Reaktivní metoda**:
        - **Třída `ReactiveAutoscaler`**: V pravidelných intervalech kontroluje průměrnou zátěž aktivních kontejnerů a přidává nebo odstraňuje kontejnery podle prahových hodnot (`SCALE_UP_LOAD` a `SCALE_DOWN_LOAD`).
    - **Prediktivní metoda**:
        - **Třída `PredictiveAutoscaler`**: Predikuje maximální počet požadavků za interval a vypočítává potřebný počet kontejnerů na základě požadované zátěže a latence.

Metody škálování

- **Výpočet počtu kontejnerů**:

    - **Reaktivní škálování:**

      $$Load_{average} = \frac{Total\ Load}{Active\ Containers}$$

      Pokud $Load_{average}$ překročí $SCALE\_UP\_LOAD$, přidají se kontejnery, naopak při poklesu pod $SCALE\_DOWN\_LOAD$ se odstraní.

    - **Prediktivní škálování:**

      $$N_{containers} = \frac{R_{predicted} \cdot t_{response}}{Load_{desired}}$$

      Predikuje se vždy dopředu o dobu, do níž nebude možno provést další predikci a spustit/odebrat další kontejnery.

Statistiky

- **Konceptuální model**: Statistické výstupy zahrnují dobu odezvy, porušení SLA a náklady na provoz.
- **Implementace**:
    - **Knihovna Simlib** poskytuje třídy `Stat` a `Histogram`, které ukládají data a generují výstupy.
    - Statistika doby odezvy je implementována jako instance `Stat response_time_stat` a `Histogram response_time_hist`.

4.2 Mapování abstraktního modelu na implementaci

Konceptuální prvek	Třída/metoda v implementaci
Požadavky	*`Request, RequestGenerator`*
Kontejnery	*`Request, RequestGenerator`*
Reaktivní škálování	*`ReactiveAutoscaler`*
Prediktivní škálování	*`PredictiveAutoscaler`*
Generování reálné zátěže	*`GenerateRealRequestsPerMinute`*
Statistika doby odezvy	*`GeneratePredictedLoad`*
Náklady na provoz	Výpočet v hlavní funkci *`main`*

5. Podstata simulačních experimentů a jejich průběh

6. Shrnutí simulačních experimentů a závěr

7. Bibliografie

[1] ALHARTHI, Saleha; ALSHAMSI, Afra; ALSEIARI, Anoud; ALWARAFY Abdulmalik; 2024. Auto-Scaling Techniques in Cloud Computing: Issues and Research Directions. Dostupné z: https://www.mdpi.com/1424-8220/24/17/5551. [cit. 2024-11-30].

[2] CLOUD NATIVE COMPUTING FOUNDATION; 2024. Pod Lifecycle. Dostupné z: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/. [cit. 2024-11-30].

[3] CONRAN, Matt; 2015. Load Balancing and Scale-Out Architectures. Dostupné z: https://network-insight.net/2015/02/26/load-balancing-and-scale-out-architectures/. [cit. 2024-11-29].

[4] DANZIG, Peter; et al. 2008. The Internet Traffic Archive. Dostupné z: https://ita.ee.lbl.gov/html/traces.html. [cit. 2024-11-29].

[5] Chris Jones, John Wilkes, and Niall Murphy with Cody Smith; 2017. Service Level Objectives. Dostupné z: https://sre.google/sre-book/service-level-objectives/. [cit. 2024-11-30].

[6] NEWMAN, David; 2024. Load Balancing Fundamentals: How Load Balancers Work. Dostupné z: https://www.linode.com/docs/guides/load-balancing-fundamentals/. [cit. 2024-11-27].

[7] PERINGER, Petr; MARTINEK, David; LEŠKA, David; 2021. SIMLIB/C++ Documentation. Dostupné z: https://www.fit.vut.cz/person/peringer/public/SIMLIB/doc/html/. [cit. 2024-11-26].

[8] PERINGER, Petr; 2022. Modelování a simulace - Studijní opora. Dostupné z: https://moodle.vut.cz/pluginfile.php/900581/mod_resource/content/2/opora-ims.pdf. [cit. 2024-11-28].

[9] PERINGER, Petr; HRUBÝ Martin; 2024. Modelování a simulace - Prezentace. Dostupné z: https://www.fit.vut.cz/person/peringer/public/IMS/prednasky/IMS.pdf. [cit. 2024-11-26].

[10] RABIU, Shamsuddeen; CHAN, Huah Yong; 2022. A Cloud-Based Container Microservices: A Review on Load-Balancing and Auto-Scaling Issues. Dostupné z: https://www.researchgate.net/publication/366562529_A_Cloud-Based_Container_Microservices_A_Review_on_Load-Balancing_and_Auto-Scaling_Issues. [cit. 2024-11-27].

[11] MICROSOFT - AzureDevOps; 2000. Microservices architecture design. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/microservices/. [cit. 2024-11-29].

[12] NIST; Technical Series Publications. Dostupné z: https://nvlpubs.nist.gov/. [cit. 2024-11-30].

[13] SAFA, Bouguezz; 2024. Název: Podnázev. Dostupné z: https://www.baeldung.com/cs/scaling-horizontally-vertically. [cit. yyyy-mm-dd].

[14] BARROSO, Luiz André; HÖLZLE, Urs; RANGANATHAN, Parthasarathy; 2019. The datacenter as a computer. Dostupné z: https://link.springer.com/book/10.1007/978-3-031-01761-2. [cit. 2024-11-29].

VZOROVÁ BIBLIOGRAFIE:

1 autor:

[15] PŘÍJMENÍ Jméno; 2000. Název: Podnázev. Dostupné z: link.example.com. [cit. yyyy-mm-dd].

N autorů:

[16] PŘÍJMENÍ Jméno; PŘÍJMENÍ Jméno; PŘÍJMENÍ Jméno; 2000. Název: Podnázev. Dostupné z: link.example.com. [cit. yyyy-mm-dd].

Mnoho autorů:

[17] PŘÍJMENÍ Jméno; et al. 2000. Název: Podnázev. Dostupné z: link.example.com. [cit. yyyy-mm-dd]