ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kovařovic** Jméno: **Karel** Osobní číslo: **406907**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vizualizace procesorového a paměťového zatížení cloudu

Název diplomové práce anglicky:

Visualization of cloud computational power and memory consumption

Pokyny pro vypracování:

Analyzujte techniky pro vizualizaci statických a v čase proměnných hierarchických skalárních dat. Dále analyzujte možnosti aplikace těchto technik k vizualizaci procesorového a paměťového zatížení cloudu. Na základě analýzy navrhněte a implementujte aplikaci umožňující monitoring procesorového a paměťového zatížení cloudu a analýzu procesorového a paměťového zatížení v určitém časovém intervalu. Dále se zaměřte na přehlednou vizualizaci relace mezi servery a virtuálními servery (např. kolik paměti či výkonu serveru používá daný virtuální server). Aplikaci otestujte na pěti datových sadách rozdílné složitosti získaných z reálného cloudu (nebo z jeho simulace) sestávajícího alespoň z deseti serverů a třiceti virtuálních serverů. Cílem testování je vyhodnotit časovou a paměťovou náročnost implementované aplikace v závislosti na složitosti dat a demonstrovat škálovatelnost navržené vizualizace, tedy pro jak složitá data je vizualizace stále použitelná.

Seznam doporučené literatury:

[1] Aigner, W., Miksch, S., Müller, W., Schumann, H. and Tominski, C. Visualizing time-oriented data - a systematic view. Computers & Graphics, 31(3), 401-409, 2007.
[2] Munzner, T. Visualization analysis and design. CRC Press, 2014.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky

Diplomová práce

Vizualizace procesorového a paměťového zatížení serverového cloudu

Bc. Karel Kovařovic

Vedoucí práce: Ing. Ladislav Čmolík, Ph.D.

24. května 2018

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Ing. Ladislavu Čmolíkovi, Ph.D. za vytrvalou výpomoc a profesionální vedení prostřednictvím průběžných konzultací. Rovněž bych chtěl poděkovat Ing. Jiřímu Chludilovi za pomoc ze strany cloudového systému BigCloud.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 24. května 2018 .....

České vysoké učení technické v Praze
Fakulta elektrotechnická
© 2018 Karel Kovařovic. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě elektrotechnické. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Kovařovic, Karel. *Vizualizace procesorového a paměťového zatížení serverového cloudu*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2018.

Abstrakt

Práce se věnuje vizualizaci serverového cloudu, konkrétně vizualizaci paměťového a procesorového zatížení jednotlivých virtuálních strojů ve vztahu k vlastnímu serveru. Za tímto účelem je nejprve podrobně analyzována problematika volby vhodných vizualizačních technik vzhledem k případům užití a to tak, aby se co možná nejvíce minimalizovalo množství práce vyžadované od uživatele při vlastní správě serverového cloudu. Výsledky této analýzy jsou poté zohledněny při návrhu a vytvoření prototypu, na jehož základě je následně implementována finální aplikace.

Klíčová slova

paměťové zatížení, procesorové zatížení, serverový cloud, virtuální stroj, vizualizace dat, grafický návrh, vizualizační technika, monitoring, sledování trendů, časově proměnná data

Abstract

The thesis deals with the problem of visualization of server cloud, namely visualization of memory and processor load of individual virtual machines in relation to the specified server. To this end the issue of selecting appropriate visualization techniques with regard to use cases is analyzed in detail, in order to minimize the amount of work required from the user when managing the server cloud as much as possible. The results of this analysis are then taken into account in the design and creation of the prototype, on the basis of which the final application is subsequently implemented.

Keywords

memory usage, processor usage, server cloud, virtual machine, data visualisation, graphical design, visualisation technique, monitoring, trend development, time-varying data

Obsah

Úvod

Vizualizace dat a její význam v současnosti

Snaha o zjednodušení vnímání, a tím pádem i pochopení většího či menšího množství informací, provázela lidstvo prakticky od nepaměti. Jedním z hlavních nástrojů byly vždy rozličné způsoby vizuální výpomoci (alternativně vizuální komunikace), které umožňovaly jak konkretizovat abstraktní tématiku, tak rozšiřovaly perceptuální vnímání člověka a celkově zlepšovaly orientaci v problému. Od původně primitivních snah, které se týkaly primárně snížení nároků na lidskou paměť v rámci nejrůznějších obchodů či případné taktické vizualizace vojenských tažení však s postupem času společně s rozvojem lidského vědění přibylo nespočet oblastí života, jejíchž se vizualizace stala takřka nezbytnou a nenahraditelnou součástí. Skutečně pravá renesance pro vizualizaci však nastala až v průběhu 20. století, a to hlavně díky nástupu a masivnímu rozšíření výpočetní techniky. Ta totiž umožňovala zpracovávání do té doby nepředstavitelného kvanta informací ve velmi krátké době. Aby byl člověk schopen schopen z takovéhoto množství dat získat pro něj relevantní informace, bylo zapotřebí rozvinutí vizualizace jako důležité vědní disciplíny, která pomocí nejrůznějších technik usnadňuje rozpoznávání důležitých vzorů a vyhledávaných souvislostí - ***Vizualizace dat***.

Moderní vizualizace dat byla původně spojena především s vědeckou obcí a nejrůznějšími výzkumy, a to zejména z důvodu počáteční nedostupnosti výpočetní techniky pro běžné uživatele. Rovněž techniky vizualizace dat byly obecně používány zejména při zpracovávání velmi velkého množství dat, jejichž zdrojem byl typicky výzkum (data ze senzorů apod.). S postupem času se však počítače rozšířily i za hranice vědeckých institutů a laboratoří a získaly další druh využití - komerční. Počítače tak začaly být zprostředkovatelem rozmanitých služeb a staly se nedílnou součástí nejrůznějších systémů. Jedním z takových systémů mohou být i cloudové servery, tedy forma poskytování serverových služeb uživatelům za pomocí distribuovaného systému, které se jako

služba začaly dramaticky rozšiřovat s počátkem 21. století. Spravování takovýchto rozsáhlých uspořádání pak vyžaduje poměrně značné úsilí, které však může vhodné využití patřičných vizualizačních technik značně snížit. Z tohoto důvodu je v současné době vizualizace dat a postupy s ní spojené důležitou součástí při správě (nejen) takovýchto systémů.

Cíl práce

Hlavním cílem této diplomové práce je na základě uživatelských požadavků vytvořit nástroj, který prostřednictvím vhodně zvolených vizualizačních technik umožní jednoduché řešení problematik spojených s monitoringem serverového cloudu, jednotlivých serverů a virtuálních strojů z hlediska paměťové a procesorové zatíženosti těchto systémů. Typ cloudu, pro který bude tento nástroj využit, je BigCloud, avšak do budoucna by bylo vhodné rozšířit použitelnost aplikace i na další druhy cloudových infrastruktur.

Za účelem vytvoření zmíněného nástroje je v rámci této diplomové práce nutné vyhotovit podrobnou analýzu stran vlastních vizualizačních technik, jejich výhod a nevýhod, jejich možných kombinací a použitelnost pro jednotlivé případy užití s přihlédnutím k typu a formátu vizualizovaných dat. Na základě této analýzy potom vybereme nejlepší kandidáty, pro které vytvoříme příslušný prototyp za účelem ověření použitelnosti těchto technik, respektive jejich možných nedostatků. Tento prototyp se tedy bude soustředit výhradně na použitelnost konkrétně vybraných vizualizačních technik s předpřipravenými umělými daty.

Na základě tohoto postupu poté bude následovat implementace finální aplikace, kde budou zohledněny nedostatky, jež vyplynou z vytvořeného prototypu a v němž budou zahrnuty případné úpravy ze strany vybraných technik a postupů. Finální aplikace bude rovněž již pracovat s reálnými daty, v krajním případě s generátorem dat, jehož výstup bude podobnostně odpovídat skutečným datům ze serverů. Bude se tedy rovněž zabývat sběrem dat, jejich uložením a případnými dalšími úpravami.

V závěrečné části poté budeme vytvořený nástroj vyhodnocovat ve vztahu k případům užití, a to zejména prostřednictvím různých datových sad s rozlišnými rozsahy. Rovněž se pokusíme o základní uživatelské testování, které nám umožní získat dodatečné informace o použitelnosti a případných nedostatcích celkové aplikace. Vyhodnocovat budeme rovněž její vlastní časovou a paměťovou náročnost. V neposlední řadě zmíníme možnosti vylepšení do budoucna a případná další rozšíření funkcionalit jako takových.

Případy užití

Případy užití primárně určují, co bude uživatel chtít s aplikací dělat, respektive jaké činnosti bude na serverovém cloudu sledovat a jakým způsobem by mu

měl výsledný nástroj v tomto směru pomoci. Předně mezi ně patří následující:

- **Monitoring**
  Uživatel dlouhodobě vizuálně monitoruje zatížení serverů a virtuálních strojů a má celkový přehled nad tím, co se v rámci cloudem děje

- **Detekce abnormalit**
  Uživatel na základe vizualizace včas detekuje možné abnormality a útoky na cloudový systém a detekuje, který server, popřípadě virtuální stroj, je napaden či jinak nezvykle vytížen

- **Srovnávání vývoje trendů v čase**
  Uživatel porovnává vývoje zatíženosti serverů a virtuálních strojů v rámci daného časové období, tj. hodin, dnů, týdny apod.

V rámci vývoje výsledné aplikace je tedy nezbytné se primárně soustředit na tyto zmíněné případy užití a následně vhodně zvolit kombinaci takových vizualizačních technik, jejichž použitím dosáhneme pokud možno co nejlepších výsledků (zejména časových) stran ulehčení daných činností.

Rozbor zadání práce

V této části jsou rozebrány konkrétní úkoly ze zadání diplomové práce.

Analyzujte následující problematiku

- **Techniky pro vizualizaci statických a v čase proměnných hierarchických skalárních dat**
  Pro různé typy dat je obecně vhodné použít různé vizualizační techniky, které příslušným způsobem s nimi ulehčí práci (ve vztahu ke konkrétní činnosti). Naším úkolem je tedy analyzovat možné varianty a následně zvolit takové techniky, jež pokud možno nejlepším způsobem umožní orientaci v našich datech, které se skládají typově zejména z dat statických (konkrétní okamžik, konkrétní míra zatížení) a časově proměnných dat (v daném časovém rozsahu, vývoj zatížení přes určitý interval, v určitých periodách apod.). Všechny tyto údaje jsou poté daty skalárními, tedy lze je vyjádřit jednou konkrétní číselnou hodnotou v daných jednotkách.

- **Možnosti aplikace zvolených technik k vizualizaci procesorového a paměťového zatížení cloudu**
  Na základě předchozí analýzy je třeba zvážit použitelnost vybraných technik ve vztahu k naší problematice, tedy procesorovému a paměťovému zatížení cloudu. Tento úkol se zabývá především rozboru použitelnosti jednotlivých postupů v kombinaci s případy užití, jejich potřebnou úpravu, případně zvolené kombinace konkrétních technik.

Navrhněte aplikaci umožňující monitoring a analýzu procesorového a paměťového zatížení cloudu v určitém časovém intervalu

Ve vztahu k informacím získaných z analýzy v bodě 1 bude předmětem této části navrhnout příslušná řešení ve formě aplikace. Součástí návrhu bude i implementace výchozího prototypu, na jehož základě dojde k posouzení použitelnosti a nedostatků vybraných technik tak, aby tyto poznatky mohli být zohledněny v případě finální aplikace. Kromě monitoringu a uživatelské analýzy procesorového a paměťového zatížení se rovněž zaměříme na vizualizaci relace mezi vlastními servery a konkrétními virtuálními stroji.

Na základě návrhu a vytvořeného prototypu implementujte vybrané postupy do finální aplikace

V této části popíšeme zvolené postupy při vlastní implementaci, včetně architektury vytvořené aplikace a jejich jednotlivých částí a práce s vlastními zdrojovými daty.

Aplikaci otestujte na pěti datových sadách rozdílné složitosti získaných z reálného cloudu (nebo z jeho simulace) sestávajícího alespoň z deseti serverů a třiceti virtuálních serverů

Výslednou aplikaci v této části podrobíme důkladnému testování za účelem zjištění použitelnosti ve vztahu k případům užití, přičemž rozsah vybraných dat umožní odhalení dalších případných nedostatků, které jsme v rámci prototypu nezaznamenali. Cílem testování je rovněž demonstrovat škálovatelnost navržené vizualizace, tedy pro jak složitá data je vybraná vizualizace stále použitelná a tedy kde jsou její limity.

Struktura textu

Text následující diplomové práce je v korespondenci s jednotlivými částmi zadání (viz. výše) rozdělen do pěti kapitol, z nichž každá se podrobněji věnuje řešení zadaných úkolů.

Analýze obecných vlastností vizualizace a následnému rozboru jednotlivých technik s přechodem do analýzy existujících řešení se podrobně věnuje kapitola ***Analýza*** 1. Následuje kapitola ***Návrh*** 2, která popisuje na základě předchozí analýzy vytvořené návrhy dílčích částí aplikace, ať už ze strany architektury systému, GUI jednotlivých pohledů či popisu vybraných algoritmů. V kapitole ***Implementace*** 3 je stručněji popsána výsledná implementace vizualizační aplikace včetně obecného popisu jednotlivých tříd a vybraných metod, z nichž některé implementují zmíněné algoritmy z přecházející kapitoly.

Kapitola ***Výsledky*** 4 dále obsahuje krátký popis jednotlivých částí výsledné aplikace a její funkcionality. V poslední kapitole ***Testování*** 5 jsou poté na závěr uvedeny výsledky testování vytvořené aplikace, kde bylo hlavním cílem ověření kvality řešení, ať už ze strany výkonu či celkové použitelnosti.

Kapitola 1

Analýza

Kapitola analýza se podrobně věnuje rozboru jednotlivých problematik a snaží se vytvořit si komplexní obraz o možných výhodách a úskalích, která mohou nastat v případě použití jednotlivých vizualizačních technik a postupů s nimi spojených.

1.1 Historie vizualizace dat

Abychom mohli lépe proniknout do významu moderní vizualizace dat a pochopit tak možnosti této vědecké oblasti, je nejprve vhodné se podívat na postupný vývoj vizualizace jako takové v průběhu historie.

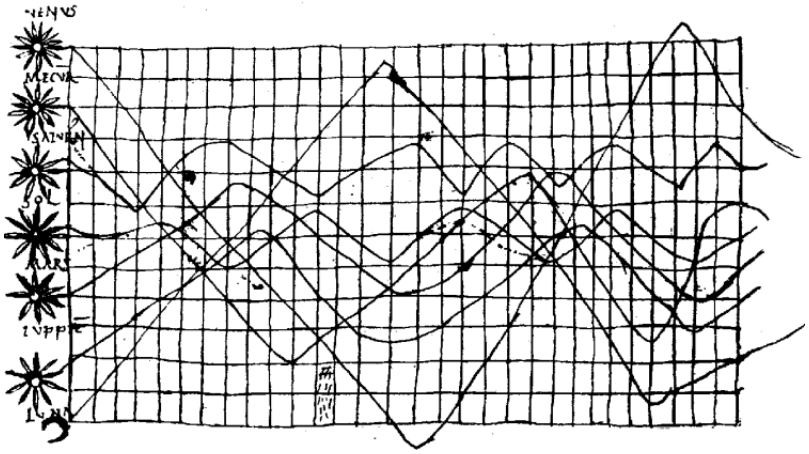
1.1.1 Počátky vizualizace

Navzdory obecnému přesvědčení, že vizualizace dat je záležitostí striktně moderní, která je stará nanejvýše několik desetiletí, vizualizace jako taková provází lidstvo už od jeho vzniku. V počátcích se jednalo především o data spojená s hvězdnou oblohou, kupříkladu informace o poloze hvězd byly ne zřídkakdy zakreslovány na stěny jeskyní již v dobách pleistocénu (jeskyně Lascaux v jižní Francii), a to zejména za účelem výpomoci při navigaci a průzkumu. Rovněž některé dochované artefakty z Mezopotámie (5500 př.n.l.), či konkrétně Kipu z Incké říše (2600 př.n.l.), lze považovat za prvotní formy vizualizace. Velmi důležitou je v tomto směru Turínský mapový papyrus z 12. století před naším letopočtem, který znázorňuje rozložení geologických zdrojů včetně informací o jejich těžbě. Tato mapa je tedy příkladem tzv. tématické kartografie, která zahrnuje mapy s konkrétní specifickou tématikou společně s přidruženými informacemi.

O něco později se objevuje prvotní myšlenka souřadnic, a to v Egyptě circa 200 let před naším letopočtem, kde byly naivní souřadnicové systémy použity při stavbě měst. Rovněž zakreslené polohy objektů jak na Zemi, tak na obloze, byly vztaženy k systému ne nepodobnému zeměpisné délce a šířce. Z období

kolem počátku našeho letopočtu následně pochází i jedna z prvních mapových projekcí sférické Země, jejíž autorem je řecký geograf a matematik Klaudios Ptolemaios. Tato projekce byla poté dlouhou dobu považována za standard, a to až do 14. století našeho letopočtu.

Významnějším dílem, které v současnosti řadíme mezi jednu z nejčasnějších vizualizací kvantitativní informace, je bez pochyby graf 1.1 vícenásobných časových řad z 10. či 11. století, který znázorňuje pohyb sedmi nejvýznamnějších nebeských útvarů, tedy planet, a to nejen vzhledem k prostoru, ale zároveň i k času. Svislá osa v grafu představuje sklon oběžných drah, zatímco horizontální osa zobrazuje čas rozdělený do třiceti intervalů. Z grafu je rovněž patrné použití sinusoid s různými periodami a taktéž souřadnicové mřížky, což může implikovat znalost elementárního souřadnicového systému, který se obecně začal používat až v průběhu 17. a 18. století.



Obrázek 1.1: Planetární pohyby zobrazované jako cyklické sklony v závislosti na čase [1]

Ve 14. století se poté objevila myšlenka vykreslování matematických funkcí do grafů (sloupcový graf) a později v témže století v díle biskupa Mikuláše Oresmeho (1323-1382) je poprvé použit postup zápisu hodnot do tabulek a následná vizualizace logických vztahů mezi těmito hodnotami. Rovněž se dochoval graf z počátku 15. století, jež znázorňuje teoretický vztah mezi rychlostí a vzdáleností a jehož autorem je německý matematik, kardinál Mikuláš Kusánský (1401-1464).

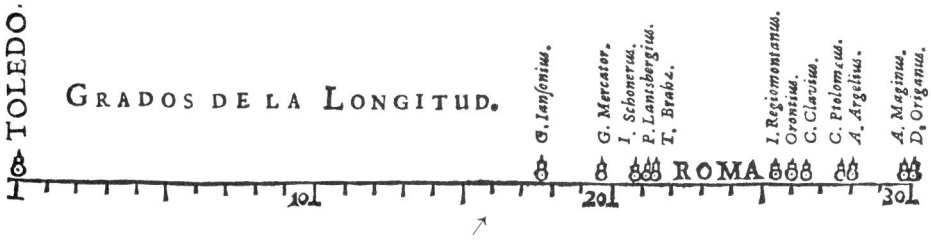
1.1.2 Vizualizace v novověku

S příchodem novověku se významným způsobem rozvinuly nejrůznější postupy a metodiky pro přesné pozorování a měření fyzikálních veličin, jakožto i poloh

geografických a astronomických objektů. Nejčastěji je v tomto směru citována například metoda zedního kvadrantu (angl. Mural quadrant) používána Tycho de Brahem, která umožňovala poměrně přesné mapování oblohy. Díky lepší přesnosti těchto postupů nabyla vizualizace jako taková na významnosti. Důležitým posunem v tomto směru představovala i nově používaná triangulace a obdobné metody pro přesné určení polohy. V roce 1545 pak holandský astronom Regnie Gemma Frisius použil tzv. cameru obscuru k pozorování zatmění slunce. Výsledný obrázek lze tedy považovat za jednu z prvních snah o přímé zachycování obrazu jako takového. Následný rozvoj nejrůznějších technik pro záznam matematických funkcí, zejména v tabulkách, stejně tak jako vydání prvního moderního kartografického atlasu v roce 1570 pouze dále podnítily všeobecný zájem o co možná nejlepší obecnou vizualizaci dat.

S příchodem 17. století se začala významně rozvíjet problematika zabývající se fyzickým měřením, a to ať už času, vzdálenosti nebo prostoru - opět primárně za účelem podrobnějšího astronomického výzkumu, popřípadě tvorby navigačních map v rámci nastupující kolonizace. Nejdůležitější pro vizualizaci byl však všeobecný růst teoretických znalostí a s ní spojený nástup praktických aplikací. Jednalo se primárně o analytickou geometrii a souřadnou soustavu (Descartes a Fermat), teorie chybovosti měření a odhadů (Galileo), počátky pravděpodobnosti (Pascal a Fermat), demografická statistika (Graunt) a další demograficko-ekonomické studie (nárůst populace, daně, pozemky, cena zboží atd.).

Významným příkladem vizualizace z tohoto období, který je v současné době považován za prvotní vizuální reprezentaci statistických dat, je lineární graf 1.2 nizozemského astronoma Michaela van Langrena z roku 1644, jež znázorňuje změnu zeměpisné délky mezi Toledem a Římem. V rámci pravdivosti je však nutné zmínit, že výsledek není zcela správný, liší se circa o 7 stupňů.



Obrázek 1.2: Langrenův graf určení zeměpisné délky od Toleda do Říma [2]

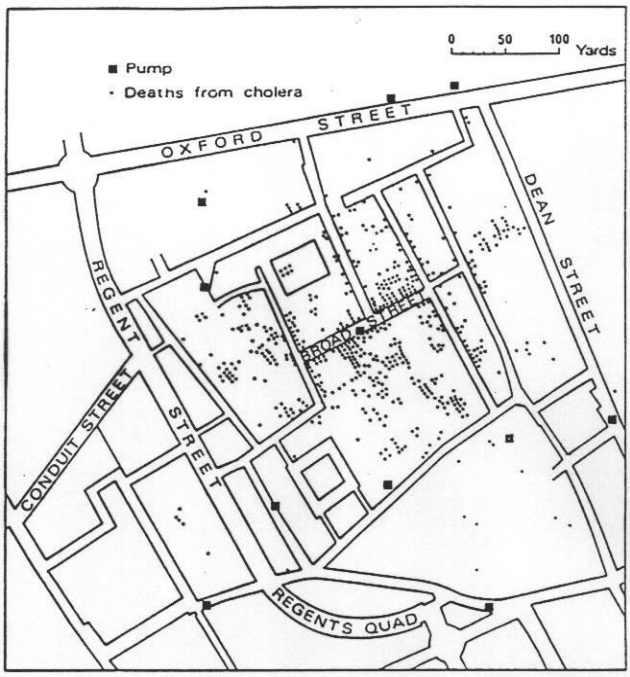
Ke konci 17. století byly tedy obecně k dispozici nezbytné prvky pro vývoj modernějších vizualizačních metod; předně tedy reálná data, která měla určitý smysl a bylo vhodné je zkoumat, dále teoretický základ, který umožňoval pochopit význam jednotlivých aspektů těchto dat a v neposlední řadě rovněž i nové ideje stran vizuální reprezentace a vizualizačních technik.

18. století bylo díky rozvoji matematických teorií v předchozím století svědkem nevídané expanze grafických reprezentací do dosud netušených rozměrů a forem. Velký rozvoj zaznamenala především kartografie, která se nově nesoustředila pouze na zaznamenání geografické pozice, ale například i na tématické mapování fyzikálních či demografických veličin. Za tímto účelem byla vytvořena nová reprezentace dat, konkrétně kontury a linie. S postupným vývojem empirického sběru dat pak došlo k rozšíření abstraktních grafů a grafů nejrůznějších funkcí.

Historicky důležitou osobností stran vizualizace dat byl William Playfair (1759 - 1823), skotský ekonom, který je považován za objevitele několika vizualizačních technik, které používáme i v současnosti. Předně se jedná o liniový graf, sloupcový graf a později dokonce i kruhový diagram. Zajímavostí z konce 18. století pak může být fakt, že stále častější využívání souřadných soustav pro vizualizaci v rámci nejrůznějších vědních oborů přimělo londýnského doktora Buxtona k tomu, aby si v roce 1795 patentoval čtverečkovaný papír.

První polovina 19. století se díky inovacím ze strany návrhu a vizualizačních technik dočkala masového rozšíření statistických grafů a tématického mapování. Pro vizualizaci statistických dat byla již touto dobou hojně používána drtivá většina moderních forem zobrazení: sloupcový graf, kruhový diagram, histogramy, liniové grafy, grafy časových řad, kontury a další. V rámci tématické kartografie poté začaly hojně vznikat naučné atlasy, které obsahovaly mapy se zaměřením na nejrůznější tématiku, ať už ekonomii, počasí, medicinu či nejrůznější sociální aspekty. Vědecké publikace byly poté přímo zahlceny nejrůznějšími grafy a dalšími vizualizacemi.

Velmi významným a všeobecně rozšířeným případem vědecké vizualizace 19. století je slavná mapa 1.3 doktora Johna Snowa vyobrazující oběti epidemie cholery v Londýně roku 1855. Původce této choroby, kterým byla znečištěná voda, byl tehdy ještě neznámý. Právě mapa doktora Snowa, kde byla jednotlivá úmrtí zaznamenána do mapy ve formě teček, pomohla tohoto původce objasnit. Tečky se totiž až nápadně shlukovaly v oblasti Broad Street, kde byla veřejná vodní pumpa. Obdobný postup zvolil již v roce 1833 doktor Robert Baker při epidemii v Leedsu. Úspěch této metody tedy významně pomohl k ustanovení vizualizace jako solidního vědeckého mechanismu i v rámci mezilidských (sociálních, lékařských, etnických) témat a ustanovil pozici doktora Snowa jakožto zakladatele moderní epidemiologie.



Obrázek 1.3: Mapa Johna Snowa zobrazující oběti epidemie cholery v Lodýně roku 1855 [3]

Druhá polovina 19. století je poté právem označována za zlatý věk statistické vizualizace. Státní úřady po celé Evropě hojně používaly vizualizaci pro plánování pokračující výstavby a industrializaci. Statistické teorie, se kterými přišel v největší míře Gauss a Laplace, a které byly následně rozšiřovány do sociální domény André-Michelem Guerrym a Adolfem Queteletem, umožnily pochopit informace spojené s velkým množstvím dat. Objevily se první vizualizace 3D povrchů za pomocí kontur, Charles Minard uveřejnil své slavné grafy zobrazující vývoj Napoleonova tažení na Moskvu roku 1812. Rovněž měřítka a tvary grafů se významně obměňovaly, od semi-logaritmických grafů zobrazujících procentuální změnu v ceně komodit v rámci časového intervalu, po nomogramy s použitím několika rovnoběžných os. Oficiální vládní statistiky byly také často vydávány pro veřejnost jakožto statistické atlasy obsahující nejrůznější formy vizualizace dat převážně spojených s populací tamní země. Obzvláště významnou roli v tomto směru hrála Francie, která každoročně vydávala atlas *Albums de Statistique Graphique*. S příchodem nového století však byla produkce tohoto atlasu kvůli vysokým nákladům ukončena, což příznačně předznamenalo velkou krizi vizualizace jako takové v počátcích 20. století.

1.1.3 20. století až současnost

Počátek 20.století byl ve vztahu k vizualizaci v pravdě dobou temna. Vizualizace se stala spíše zdrojem pěkných obrázků, než solidní vědeckou disciplínou, a bylo na ní možná nahlíženo i s jistým despektem. Tato situace byla zapříčiněna především obdobím kolem první světové války, kdy byla místo inovace důležitá především popularizace. Díky stále zpřesňujícím se výpočtům byla čísla přesnější a nevyžadovala pomocnou vizualizaci v té míře, jako v předcházejícím století, respektive s čím dál větší přesností dat bylo stále obtížnější menší a menší rozdíly správně vyobrazovat. Na druhou stranu se však obecné grafy a statistické vizualizace staly nedílnou součástí výukových materiálů na školách i zpráv státních a komerčních úředníků.

Důležitým dílem pro vizualizaci počátku 20. století se stala kniha sira Arthura Bowleyho *Elements of Statistics*. V několika kapitolách zde autor popisuje využití grafů a diagramů a vysvětluje grafické metody spojené s frekvenčními křivkami a hledáním mediánu a kvartilů. Rovněž se zabývá efekty spojenými se změnou měřítka, interpolací hodnot či obdélníkovými diagramy, které umožňují zobrazit tři nezávislé proměnné prostřednictvím výšky, délky a hloubky. Zajímavostí pak mohou být "historické diagramy", ve kterém mohou být zobrazeny dvě nebo i více časových řad za účelem jejich vzájemného porovnávání.

Navzdory počáteční nedůtklivosti 20. století vůči vizualizaci se tento vědní obor dočkal ještě v rámci téhož století masového rozšíření do takřka všech oblastí vědeckého, komerčního i každodenního života. K tomu přispěl hlavní měrou vývoj na tu dobu nesmírně výkonných strojů pro statistické zpracovávání dat a především pak přirozeně příchod výpočetní techniky jako takové v doprovodu s nejrůznějšími zobrazovacími zařízeními. Tento vědecký pokrok tedy započal v pravdě novou renesanci pro vizualizaci dat, a to zejména v období šedesátých let 20. století. V této době se událo několik pro grafiku velmi významných událostí; V roce 1961 vyšel důležitý vědecký článek *The Future of Data Analysis* od Johna Tukeyho, který zdůrazňoval význam grafické datové analýzy. Dále pak v roce 1967 francouzský kartograf Jacques Bertin vydal knihu *Sémiologie Graphique*, jejíž obsahem byla mj. i myšlenka organizace vizuálních a perceptuálních prvků vizualizace dle vlastností a vztahů v datech. V neposlední řadě byl pro vizualizaci velmi podstatný i programovací jazyk FORTRAN, který umožnil vznik několika zásadních grafických programů. Díky těmto třem zásadním milníkům se objevilo mnoho nových postupů, metodik a vizualizačních technik, mezi kterými převládaly nové způsoby zobrazování vícerozměrných dat - např. Chernoffovy tváře1.4, paprskový graf či některé stromové reprezentace. Díky možnostem výpočetní techniky pak na důležitosti postupně nabrala i lidská interakce s vizualizací samotnou.

Obrázek 1.4: Chernoffovy tváře [18]

V současné době je vizualizace dat považována za velmi seriozní a významný vědní obor. Je však poměrně obtížné jednotným způsobem popsat její současný vývoj, a to zejména díky obrovskému množství technik a způsobů, které vizualizace obsahuje a kterými se zabývá. Přesto lze určit několik základní směrů a témat, na které se vizualizace jako taková aktuálně soustředí. Předně se jedná o vizualizaci tzv. big dat, tedy dat obrovských rozměrů, které nutně vyžadují právě za účelem lepší orientace v nich určitou formu abstraktní vizualizace. Rovněž důraz na kognitivní a perceptuální aspekty jednotlivých vizualizačních technik je důležitým aspektem současného výzkumu, jakožto i hledání nových způsobů pro zobrazování vysoko-dimenzionálních informací či práce s dynamickými grafy. Důraz je kladen taktéž na uživatelovu interakci, která musí být intuitivní a neměla by vynucovat uživatelovu podrobnější znalost o vizualizačním systému či o technikách samotných.

Všechny tyto trendy současné vizualizace jsou pochopitelně do značné míry závislé na technologickém pokroku, a to v pravdě více než kdy jindy. S rostoucími schopnostmi výpočetní a zobrazovací techniky je možné zpracovávat stále více dat a tyto tedy i vizualizovat s nejrůznějšími cíli a nejrůznějšími způsoby. Abychom tedy správně určili jakým způsobem volit vizualizaci a které techniky vybrat, je v současnosti nezbytná podrobnější analýza na toto téma.

1.2 Důležité aspekty vizualizace

Na úvod je potřeba zmínit, že vizualizace našeho problému, tedy paměťového a procesorového zatížení serverového cloudu, je kvůli abstraktivitě těchto dat příkladem ***vizualizace informací***. Vizualizace informací obecně zobrazuje nějaká abstraktní data, která mohou být jak číselná, tak nečíselná (např. nějaký text). Na rozdíl od vědecké vizualizace, která typicky používá interaktivních vizuálních reprezentací vědeckých dat, obvykle fyzicky založených, jsou zdrojem dat pro vizualizaci informací koncepty, které jsou často v přírodě

zcela abstraktní. Účelem vizualizace informací je poté umožnit datovým analytikům, aby získali vnitřní mentální modely informačního obsahu; modely, které lze následně použít pro charakterizaci, predikci a rozhodování.

V rámci vizualizace informací, která je obecně velmi rozsáhlá, je přirozeně důležité si při výběru vhodné vizualizační techniky uvědomit, jakou formu naše data mají a s jakým cílem vizualizaci provádíme. Jinými slovy, s čím by měla koncovému uživateli pomoci, jakou práci by mu měla ulehčit, co by mu měla pomoci detekovat v krátkém čase apod. Rovněž důležitým prvkem při zvolení vhodné techniky je nepochybně charakteristika cílového uživatele. Jiné postupy budeme volit v případě zkušeného technika, který se v dané problematice orientuje, než v případě naprostého laika. Taktéž některé formy fyzické indispozice, například barvoslepost, velmi významným způsobem mění zavedené postupy (nemožnost zvolit barvu jako jeden z informačních kanálů vizualizace aj.).

Abychom správně oddělili důležité informace od redundantních v moři šumu, které nejrůznější nashromážděná data představují, je zapotřebí se důkladně zabývat jednotlivými specifickými vlastnostmi vizualizace. Mezi tyto základní vlastnosti řadíme následující:

- Typy dat
- Dimenze vizualizovaných dat
- Mapování dat
- Cíle
- Čas
- Uživatelská interakce
- Lidský faktor

V následujících odstavcích jednotlivé vlastnosti popíšeme podrobněji.

1.2.1 Typy dat

Další ze základních vlastností vizualizace je pochopitelně vlastní typ dat. Jinak budeme vizualizovat číselná data, mezi kterými je evidentní vztah (typ tohoto vztahu opět mění vizualizaci, např spojitost vs. diskrétnost), a jinak kupříkladu data vektorové podstaty v rámci dynamiky tekutin. Jeden z obecných prvních kroků při volbě vhodné vizualizace je tedy určení typu dat.

Základní kategorizace dat v závislosti na typu je následující:

- Tabulární data
  Tabulární data jsou typicky popisné informace, obvykle alfanumerické,

které jsou uloženy v řádcích a sloupcích v uspořádané databázi, kterou obecně nazýváme tabulkou. Tato data pak mohou být propojena s určitými prostorovými daty. Tabulka jako taková je pravděpodobně nejstarší datovou strukturou. Její vizualizace je v současné době téměř pro všechny zcela intuitivní a je lehce zpracovatelná moderní výpočetní technikou. Řádky tabulky představují typicky jeden objekt, sloupce poté konkrétní vlastnosti, přičemž obvykle platí, že všechny objekty mají stejnou množinu vlastností. Buňka pak představuje průnik konkrétní vlastnosti pro konkrétní objekt a obsahuje tedy příslušnou hodnotu. Pro tabulární data se typicky používá vizualizace informací.

- Relační data
  Relační data jsou v lecčems podobná datům tabulárním, s tou přidanou hodnotou, že jejich součástí je navíc vztah mezi jednotlivými tabulkami. Lze je tedy chápat jako graf, kde uzly typicky reprezentují objekty a hrany vztahy mezi nimi. Jak hrany, tak uzly mohou být spojeny s nějakými atributy. Relační data pak obvykle představují nějakou síť či hierarchii a stejně jako v případě dat tabulárních se pro data relační používá vizualizace informací.

- Prostorová data
  Prostorová data mají obecně nějakou přesně danou strukturu, která je typicky znázorňována jakožto trojrozměrná mřížka skládající se z vrcholů a buněk, popřípadě modelují nějakým obdobným způsobem reálné fenomény vyskytující se v přírodě. Jak vrcholy, tak buňky mohou být pak asociovány s určitými vlastnostmi a hodnotami. Speciálním případem prostorových dat mohou být data geoprostorová či geografická, která, jak již název napovídá, jsou data s implicitním nebo explicitním vztahem k místu na Zemi. Jelikož prostorová data reprezentují typicky reálné fyzikální veličiny, jakými může být kupříkladu teplota, tlak či rychlost, jsou k vizualizaci těchto dat použity techniky vědecké vizualizace.

- Časově orientovaná data
  Jak již název napovídá, časově orientovaná data jsou taková data, jež jsou přímo závislá na čase. Jsou tedy často příkladem některého z výše zmíněných typů dat, ke kterým je přidán čas jakožto další datová dimenze. Díky určitým charakteristikám času jsou pro tyto typy dat nezbytné specifické analytické a vizualizační metody k jejich prozkoumání a analýze. Pro detailnější analýzu časově orientovaných dat vizte kapitolu *Čas* 5.1 níže.

Tyto základní typy dat pak rovněž můžeme dále klasifikovat do dalších podkategorií na základě následujících tří metrik:

- Referenční rámec dat

- Počet proměnných
- Míra abstrakce

Pod pojmem referenčního rámce dat je myšleno to, zdali jsou vizualizovaná data abstraktní, nebo data reálná. Pro abstraktní data využíváme vizualizaci informací (v jiných zdrojích vizualizace grafová či softwarová), pro data reálná vizualizaci vědeckou (např. vizualizace proudů nebo objemová). Obecně platí, že relační a tabulární data jsou povětšinou abstraktní, prostorová data pak striktně z definice daty reálnými. Časově orientovaná data pak mohou být obojího typu.

Dle počtu proměnných pak můžeme data rozdělit do dvou skupin: univariantní či multivariantní. Univariatní data se skládají z právě jedné proměnné, v případě časově orientovaných dat z jedné časově závislé proměnné. Multivariantní data pak mají proměnných více. V případě multivariantních dat se poté objevuje nový možný cíl vizualizace - hledání korelací mezi jednotlivými proměnnými. Obecně pak platí, že čím více proměnných data obsahují, tím komplexnější vizualizační techniky je nutné aplikovat, viz. kapitola *Dimenze vizualizovaných dat* 1.2.2 níže.

Poslední pomocnou klasifikací dat je míra jejich abstrakce, čímž je myšleno to, zdali zobrazujeme skutečná data, která máme k dispozici, či je nějakým způsobem před vizualizací upravujeme, zejména za účelem snížení jejich množství či zvýraznění určitých vlastností, které jsou pro uživatele důležité. Případem abstrakce je kupříkladu agregace dat, či zvýrazňování určitých událostí v rámci časově orientovaných dat. Volba míry této abstrakce je pak tedy často závislá na přímé volbě cílů, viz. kapitola *Cíle* 1.2.4.

Je pochopitelné, že celý dataset obecně nemusí být jednoho typu. Zejména v rámci n-rozměrných dat může být část kupříkladu relační, část časově orientovaná apod. Toto obecné rozdělení tedy není vždy přesné, respektive nejrůznější kombinace jsou poměrně časté. Vlastnosti (roz. hodnoty) dat pak mohou rovněž být různého typu; nejčastěji se rozlišují podle toho, zda je lze řadit (čísla, dny v týdnu), či nikoliv (jména lidí, města).

V případě našich dat se tedy jedná o kombinaci dat časových a tabulárních (kategorických) s některými vlastnostmi relačních dat (server -> virtuální stroje), které však v zásadě v rámci našeho problému nebudeme speciálně vizualizovat. Pro více detailů stran vlastních dat vizte kapitolu Sběr a zpracování dat 1.5.

1.2.2 Dimenze vizualizovaných dat

Základní znalostí pro zvolení správné techniky je dimenze dat, které chceme vizualizovat, občas rovněž uváděna jako komplexita dat. V rámci komplexních datových setů pracujeme obecně s n-dimenionálními daty, přičemž typicky jedna proměnná = jedna dimenze. Často pak nastává problém v momentě, kdy se snažíme zobrazit n dimenzonální data do zobrazovacího prostoru, jehož

dimenze je menší než dané n, kupříkladu 3D volumetrická data na vytisknutém papíře a podobně. Z tohoto důvodu může být obtížné snažit se o vizuální prozkoumávání dat, jakmile počet rozměrů přesáhne tři. Přesto však existuje několik vizualizačních technik, které takovéto prozkoumávání v zásadě umožňují, kupříkladu metoda rovnoběžných souřadnic.

Možným řešením vysoké dimenze vizualizovaných dat může být rovněž rozdělení vlastní vizualizace do několika rozdílných pohledů (roz. grafů), z nichž každý bude zobrazovat pouze některé dimenze a jejich vztah. Je pak otázkou, zdali chceme v každém okamžiku vyobrazovat všechny tyto grafy, popřípadě do jaké míry nám stačí řešit jednotlivé dimenze samostatně.

V našem případě jsou data čtyř rozměrná - vlastní zatížení (procesor nebo paměť, konkrétní hodnota nebo procentuální ve vztahu k VM či ve vztahu k celku), čas (okamžik zaznamenání vytíženosti), konkrétní VM a server, na kterém je virtuální stroj umístěn. Mezi servery a virtuálními stroji je pochopitelně korelace, oproti tomu zatížení konkrétního serveru neovlivňuje zatížení serverů ostatních. Přesto pro uživatele bude pravděpodobně nezbytné paralelně sledovat zatížení všech serverů, a to zejména v rámci případu užití monitoringu, kdy je třeba rychle zaznamenat dramatické změny v rámci celého serverového cloudu. Vývoj dat v čase poté indikuje nutnost zvolení takových vizualizačních technik, které pracují s časově závislými daty, viz. .

1.2.2.1 Redukce dimenzionality

Zajímavým aspektem dimenzionality dat ve vztahu k její následné vizualizaci je nepochybně snaha o redukci počtu dimenzí v rámci předzpracovávání dat, a to zejména v případě skutečně vysoké výchozí dimenzionality. Tento postup přirozeně není použitelný vždy, kupříkladu pokud je potřeba paralelně pozorovat vztahy mezi všemi jednotlivými dimenzemi (i když i tuto situaci lze redukcí "obelstít", byť možná ne zcela efektivně). Studie redukce dimenzionality je následně předmětem podrobnější datové analýzy zahrnující mimo jiné i systémy strojového učení (machine learning) a rozeznávání vzorů (pattern recognition).

Vlastní redukce dimenzionality dat za účelem lepší vizualizace pak pracuje se základní myšlenkou, že n-rozměrnou informaci lze obecně vyjádřit jakožto bod v n-rozměrném prostoru. Pokud poté tento bod můžeme zobrazit do d-rozměrného podprostoru ($d < n$) prostřednictvím bezeztrátové transformace (roz. transformace při které se neztratí žádná informace z původních dat, případně je ztráta marginální nebo probíhá na dimenzích, které pro nás mají menší váhu), lze provést redukci dimenzionality poměrně jednoduše, přestože tato transformace bývá často nelineární. Příkladem může být kupříkladu sloučení několika korelovaných proměnných do faktorovaných proměnných, popřípadě redukce celého vícedimenzionálního problému do problematiky zjišťování shluků datových objektů.

Na závěr je však nutné podotknout, že čím nižší je dimenze d a čím větší

je rozdíl mezi oběma dimenzemi d a n, tím je přirozeně těžší provést tuto redukci tak, aby byla výsledná reprezentace co nejvěrnější výchozím datům, popřípadě je tato zcela nemožná. Rovněž tzv. "prokletí dimenzionality", kdy se určité vlastnosti prostorů při změně dimenzionality radikálně mění, značně znesnadňuje vlastní proces redukce. Efektivní redukce dimenzionality tak nadále zůstává předmětem výzkumu, respektive pokračuje hledání stále lepších technik pro zobrazování konkrétních případů vícedimenzionálních dat.

1.2.3 Mapování dat

V rámci vizualizace je velmi podstatným způsob, kterým budeme mapovat data, respektive jejich konkrétní kvantitativní hodnoty - v našem případě vytížení v daném časovém okamžiku/intervalu. Mapováním dat je primárně myšlen způsob, jak vizualizovat informace, respektive jak zakódovat tyto informace do vizuální podoby. V rámci mapování se data transformují do určité grafické podoby na základě jejich určitých vlastností. Dobrá forma mapování pak vytváří správné vizuální reprezentace, které jsou dosaženy za předpokladu existence přesného vztahu mezi datovými objekty a vizuálními objekty. Z tohoto důvodu může však být mapování komplikované; zatímco některá data využívají vztahů, které lze intuitivně jednoduše graficky vyobrazit, např. prostorový vztah (teploty ve městech v konkrétní zemi), mnoho datových souborů tyto "tradiční" vztahy nevyužívá.

Abychom mohli data správně namapovat, je podle autorů článku *Readings in Information Visualization: Using Vision to Think* Carda, Mackinlayeho and Sheniedermana nutná znalost následujících tří faktorů:

- Prostorová dimenze (Spatial substrate)
- Grafické prvky (Graphical elements)
- Grafické vlastnosti (Graphical properties)

Prostorová dimenze, jak již název napovídá, představuje prostor, respektive jeho dimenzi, do které budeme naše data vykreslovat. Částečně tedy koresponduje s již zmíněnou dimenzionalitou dat. V tomto ohledu se nejčastěji využívá dvoudimenzionálního karteziánského prostoru s osami X a Y, lze však využít i trojrozměrný prostor a v některých případech i hyper-dimenzionální prostor s více než třemi dimenzemi. Obecně je pak v těchto systémech jedna dimenze reprezentována jednou osou souřadnou. Každá osa může mít jiné měřítko a může být jiného typu, předně kvantitativní (předpokládá existenci nějaké metriky podle které lze data vyčíslit), ordinální (data lze seřadit do nějakého předem určeného pořadí), nebo i nominální (osa je rozdělena do úseků které mezi sebou nemají žádný vztah). V našem konkrétním případě budeme téměř exkluzivně využívat dvoudimenzionálního karteziánského prostoru s tím, že přestože naše data mají více než dvě dimenze, budeme schopni

tyto vyselektovat, respektive díky vizualizačním technikám namapovat některé dimenze jinam, než na souřadné osy. Použité osy pak budou v závislosti na konkrétních případech (roz. pohledech) jak kvantitativní (procentuální zatížení), tak nominální (konkrétní servery) i ordinální (čas - ten není kvantitativní protože neexistuje čas 0, lze však hovořit o časových intervalech).

Grafické prvky, nebo také grafické elementy, pak reprezentují všechny základní vizuální prvky, které se ve zvolené prostorové dimenzi zobrazují. Tyto elementy jsou čtyři:

- Bod
- Přímka
- Plocha
- Objem

Je poté nasnadě, že určité prvky se hodí pro vizualizaci konkrétních dat a informací a naopak pro některé jsou naprosto nevhodné. Vhodně zvolenou kombinací lze pak rovněž docílit efektu, kdy některé prvky více vystupují a jimi reprezentovaná informace tak může být uživateli více zřejmá. Jelikož v našem případě budeme pracovat s časem a jeho linearitou, je pravděpodobné, že nejčastějšími elementy kterých budeme při vizualizaci používat budou přímky, respektive plochy. Body pravděpodobně nevyužijeme kvůli jejich přílišné vizuální diskretizaci, objem poté hůře znázorňuje konkrétní kvantitativní data (zatížení).

Grafické vlastnosti představují takové vlastnosti jednotlivých základních grafických prvků, na které je velmi citlivá oční sítnice a umožňuje tak konkrétní prvky nějakým způsobem zvýraznit či naopak potlačit. Tyto vlastnosti jsou nezávislé na pozici konkrétního prvku v prostoru. Mezi nejčastěji využívané vlastnosti pak patří následující:

- Velikost
- Orientace
- Barva
- Textura
- Tvar

Kombinace těchto vlastností a elementů pak vytvářejí konkrétní vizuální podněty pro uživatele. Tyto podněty nejsou pochopitelně ekvivalentní z zhlediska schopností člověka rozeznávat kvantitativní či jinou informaci v nich zakódovanou. Pozorované rozdíly byly poté v 80. letech minulého století vědecky

ověřeny, kdy statistici William Cleveland a Robert McGill provedli několik experimentů se skupinou dobrovolníků. Na základě výsledků těchto experimentů pak bylo určeno pořadí podle toho, jak jsou tyto podněty reprezentativní stran kvantitativní informace (od nejpřesnějších po nejméně přesné):

1. Obecná pozice v prostoru (bez té prakticky nejde vizualizovat)
2. Zarovnaná délka (s určitým počátkem a daným měřítkem)
3. Nezarovnaná délka (volná, bez ukotvení)
4. Úhel, Sklon
5. Obsah, Barevná intenzita
6. Objem
7. Barevný odstín, textura

Na základě tohoto pořadí může být délka se zarovnanou osou počátku nejlepší volbou, která umožňuje lidem přesně porovnávat číselné rozdíly. To však neznamená, že je třeba se při každé vizualizaci vždy vyhnout ostatním možnostem. Barevný odstín je kupříkladu velmi dobrým způsobem jak kódovat kategorická data. Lidský mozek je poté zvláště účinný při rozpoznávání vzorů a rozdílů. To znamená, že variace v barvě, tvaru a orientaci může být dobrou volbou pro reprezentaci kategorických dat, navzdory špatné reprezentaci při kódování přesných hodnot spojitých proměnných.

Je rovněž nutné si uvědomit, že je možné (a často i nezbytné) kombinovat různé vizuální podněty k zakódování vícera proměnných v rámci jedné vizualizace. V takovém případě je potřeba dbát na to, které signály v rámci výše zmíněné hierarchie kombinujeme; některé se kombinují obtížně a výsledek tak může uživatele mást. Rovněž cíl vizualizace určuje, které podněty využít pro které informace - ty důležitější kódujeme podněty v horní části vizuální hierarchie, abychom tuto informaci nejúčinněji uživateli předali.

Barva jako taková může být obecně poměrně problematická v rámci vizualizace informací. V prvé řadě existuje mnoho kulturních, jazykových i psychologických faktorů, které mohou určitým způsobem ovlivnit vnímání konkrétních barev ve vztahu k určitým situacím, respektive jim přiřazuje specifické významy či interpretace (např. červená je v západním světě považována za varovnou barvu, na asijském východě pak často za barvu reprezentující život a štěstí).

Výrazným problémem ve vztahu k barvě je následně i barvoslepost, která postihuje až 12% mužů a 0,5% žen. Červená slepota (Protanopie) a zelená slepota (Deuteranopie) jsou v tomto směru nejčastějšími formami barvosleposti, avšak existují i jiné, kupříkladu modrá barvoslepost (Tritanopie).

Taktéž díky své kognitivní síle může použití barevného odstínu jakožto primárního rozlišovacího kanálu v rámci kategorických dat vytvářet určité problémy. Kupříkladu v případě vizualizačního přístupu Overview and Detail může stejná barva v různých kontextech implikovat vztah, který neexistuje (server stejnou barvou jako nějaký VM - spojitost). Dále je nutné si rovněž uvědomit, že za účelem co nejpřesnější a nejrychlejší identifikace rozdílnosti konkrétních kategorických dat, jež jsou rozlišeny barevným kanálem, je třeba volit odstíny, které jsou dostatečně odlišné. To se v případě většího množství položek (20+) stává velmi obtížným a je tedy třeba hledat alternativy, popřípadě barvu doplnit o další rozlišovací podnět.

V případě našich dat je kategorizace velmi častým jevem (jednotlivé VM, servery). Barevný odstín bude z tohoto důvodu opakovaně využíván. Rovněž délka, která se bude vyvíjet v čase jakožto indikátor kvantitativních hodnot, tedy vytížení procesoru a paměti v rámci konkrétního časového intervalu, bude hrát ve vizualizaci důležitou roli.

1.2.4 Cíle

Při výběru konkrétních vizualizačních technik jsou účely, za kterými vizualizaci provádíme, přirozeně hlavní rozhodovací rolí. Častým problémem u vizualizace však z tohoto hlediska bývá fakt, že nedostatky postupů zvolených pro řešení konkrétního problému nejsou typicky dopředu známy. Fáze návrhu a tvoření prototypů je díky tomu nesmírně důležitá a jako taková pomáhá často odhalit potíže spojené s danou vizualizační technikou, o kterých při jejím výběru nemáme tušení.

Mezi základní způsoby analýzy dat, které můžeme chtít nad našimi daty provádět a které nepřímo souvisí s výběrem vhodných vizualizačních technik, jež by takovouto analýzu zjednodušovaly či vůbec umožňovaly, patří následující:

- Srovnání (Comparison)
  Jak naznačuje název, tento způsob analýzy dat používáme k vyhodnocení a porovnávání hodnot mezi dvěma nebo více datovými body. V rámci srovnávání pak můžeme snadno najít nejnižší a nejvyšší hodnoty v grafu. Existuje společná podmnožina srovnávacích grafů - trendy. Grafy trendů mají obvykle časovou osu a jednu nebo více "hodnotových" os, které se používají k tomu, aby se ukázaly vývoje datasetu během určitého časového období. Typickými příklady srovnávacích vizualizačních technik může být například sloupcový či liniový graf.

- Složení (Composition)
  Myšlenkou složení je zobrazit jednotlivé části utvářející celek a to tak, že vizualizační technika kombinuje dílčí prvky a zobrazuje je jako součet. Složení lze také použít k zobrazení toho, jak lze celkovou hodnotu rozdělit na části, popřípadě zvýraznit význam každé části vzhledem k celku.

K typicky špatnému použití vizualizačních technik pro tento typ datové analýzy dochází například v případě, že jednotlivých dílčích částí je příliš mnoho nebo jsou rozdíly mezi nimi příliš velké. Tradičními typy grafů, jež zjednodušují tento druh datové analýzy, jsou například skládaný plošný graf, skládaný sloupcový graf, vodopádový graf nebo kruhový diagram.

- Rozložení (Distribution)
  Rozložení kombinuje jak srovnání, tak složení. Pomáhá zobrazovat celé datové spektrum a vizualizovat související i nesouvisející datové body. Díky vizualizačním technikám pro rozložení jsme schopni dobře hledat existující korelace v datech, trendy, vzory, clustery, průměry či odlehlé hodnoty (outliers). Velmi častá podmnožina distribuční analýzy je analýza odchylek nebo rozdílů. Tyto metody pak sledují především hodnoty, které se nějakým způsobem odchylují od standardní normy. Vizualizační techniky spojené s touto analýzou zahrnují liniové a plošné grafy, sloupcové histogramy či korelační diagram (scatter plot).

- Vztah (Relationship)
  Jak naznačuje název, grafy asociované s touto analýzou ukazují vztah, korelaci nebo spojení dvou a více objektů a jejich vlastností. Dobré využití vztahových grafů by mělo ukázat, jak některé objekty pozitivně či negativně ovlivňují nebo neovlivňují jiné objekty. Podobně jako u rozložení i vztahové grafy mohou být použity k nalezení korelace, trendů, vzorců, clusterů, průměrů nebo odlehlých hodnot. Vizualizace vztahů začleňuje kupříkladu dvoudimenzionální korelační diagramy, bublinové grafy, liniové grafy či tabulky.

V reálné situaci je samozřejmě běžné kombinování různých přístupů k analýze a tedy i volba několika vizualizačních technik, které jsou za účelem sledování konkrétních vztahu kombinovány, respektive je mezi nimi přepínáno v závislosti na požadavcích uživatele. V našem případě bude určitě nutné srovnání mezi jednotlivými VM a jejich vytížení v rámci konkrétního serveru, potažmo vytížení mezi servery samotnými, případně srovnání několika konkrétních serverů či virtuálních strojů v rámci vybraného časového období. Rovněž složení serverů ze svých VM a jejich vztah bude nutný příslušně vizualizovat, a to ať už na přímo, či vynucenou interakcí uživatele (rozkliknutí serveru -> zobrazení VM a jejich vytíženosti, respektive vytěžování serverového procesoru a paměti).

1.2.5 Čas

Zvláštní podkapitolu věnujeme času, přestože se v zásadě rovněž jedná o jednu z typových vlastností dat (Časově orientovaná data), a to zejména z toho důvodu, že naše data jsou primárně časově orientovaná a jejich správná vizualizace je tedy vztažena k možnostem jednotlivých technik ve vztahu k této

veličině. Jedním z hlavních use casů je dlouhodobý monitoring vývoje zatížení serverového cloudu v čase, kde uživatel neustále pracuje s časem a snaží se z vizualizovaných dat vypozorovat typický průběh a neobvyklé změny, popřípadě ihned detekovat konkrétní signál (roz. vysoké/nízké vytížení) pro daný časový interval/okamžik.

Vizualizace časově závislých dat je obecně netriviální záležitostí. Přestože se v současnosti vyskytuje mnoho nejrůznějších technik a metodik pro vizualizaci takovýchto dat, drtivá většina těchto postupů je úzce specifických a nelze je tedy použít pro obecný případ. Důvodem je velká rozmanitost typů dat a jednotlivých vztahů s časovou dimenzí. Tyto různé typy mohou zahrnovat kupříkladu burzovní data, data fyzikálních simulací, ale i sbírky fotografií či články internetových novin. Teoreticky by pak měl být přístup pro vizualizaci ke všem těmto typům časových dat stejný, což je však prakticky nemožné a je tedy třeba uplatnit specializované techniky - příkladem těchto metod může být např. Theme river či Ganttův diagram, viz. níže.

Přesto však existuje i několik relativně obecných vizualizačních technik, které v mnohých případech mohou tyto specializované postupy úspěšně nahradit. Tyto metody vychází z toho, že čas je všeobecně chápán jakožto kvantitativní veličina, přestože efektivně se jedná o veličinu ordinální - čas 0 v drtivé většině vizualizací nevyužíváme, taktéž aritmetické operace nad časem ne vždy dávají smysl. Příkladem může být všeobecně uznávaný postup paralelních souřadnic. Obecné metody vizualizace časově orientovaných dat však povětšinou umožňují pouze základní datovou analýzu a zřídkakdy dokáží vhodně zobrazit vztahy mezi několika typy proměnných naráz, zejména pokud je přítomno více kvantitativních proměnných. Rovněž interakce s vizualizací za účelem lepšího zkoumání dat při těchto obecných postupech nemusí být pro uživatele zcela jednoduchá.

Základním cílem, kvůli kterému provádíme vizualizaci a následnou analýzu časových dat, je primárně uživatelův zájem o vývoj těchto dat v průběhu času. Za tímto účelem je hlavním postupem porovnávání datových údajů umístěných na různých místech časové osy. Tento proces by pak zároveň měl usnadnit objevení vzorů a trendů, které následně umožňují lépe pochopit a porozumět datům jako takovým. Zároveň je však potřeba mít v potaz další specifické úkony, které uživatel s daty chce vykonávat, respektive s jakými dalšími cíli je vizualizace prováděna. Za tímto účelem je v rámci vizualizace takových dat využívána nějaká forma přímé uživatelské interakce, nejčastěji při volbě podmnožiny dat, které chceme důkladněji zkoumat (brushing). Specifikem časově orientovaných dat stran interakce je pak časová agregace, tedy způsob prohledávání časové osy, kdy je přepínáno mezi jednotlivými úrovněmi časové škály, tedy časové granularity (roz. hodiny, dny, týdny, měsíce atd.), případně jsou zvoleny specifické časové intervaly (od 10.9 do 12.9, atd.).

1.2.5.1 Kategorizace časově orientovaných dat a jejich vizualizace

Přestože jak bylo zmíněno, obecný přístup k vizualizaci časových dat je obtížný, určitá forma obecné kategorizace je dle článku *Visualizing time-oriented data—A systematic view* možná. Tato kategorizace používá jakožto výchozí body rozdělení tři základní faktory vizualizace časově orientovaných dat:

- **Kritérium času**
  Jaké jsou charakteristiky času v dané vizualizaci, jak vypadá časová osa
- **Kritérium dat**
  Jak vypadají data, která ve vizualizaci vztahujeme k časové ose
- **Kritérium reprezentace**
  Jakým způsobem tato data reprezentujeme

V rámci kritéria času je dobré zkoumat dvě základní časová primitiva z kterých se může časová osa skládat - *časové body* a *časové intervaly*. Zatímco časové body vyjadřují konkrétní časový okamžik a jsou tedy striktně diskrétní, časové intervaly jsou obecně specifikovány dvěma časovými body (začátek a konec) a mohou mít různý rozsah. Časový interval může být ovšem vyjádřen i jedním bodem s přidruženou informací o délce. Volba vhodného časového primitiva pak souvisí s platností dat a se vztahy, které mezi těmito daty existují. V rámci naší problematiky budeme využívat osu, která se skládá z jednotlivých časových bodů (milníků), a to zejména díky způsobu získávání dat, kdy data o vytíženosti souvisí s konkrétním timestampem (okamžikem). Časové intervaly jsou pak vytvářeny právě volbou dvou časových bodů, tedy začátku a konce intervalu. V rámci časové agregace pak eventuálně může dojít k vizualizaci pevně stanoveného intervalu pomocí jednoho bodu (1 den, 1 týden atd.).

Druhým důležitým prvkem v rámci kritéria času je časová struktura. Ta může být trojího typu: *lineární*, *cyklická* nebo *stromová*. Lineární struktura odpovídá intuitivnímu chápaní času - čas plyne kontinuálně a skládá se ze seřazené posloupnosti časových primitiv. Cyklická struktura pak používá konečnou množinu v čase se opakujících primitiv, např. ročních období, a nelze tedy přesně určit, jaký bod předchází kterému (v zásadě každý bod je následovníkem i předchůdcem jiného bodu). Speciální časovou strukturou je struktura stromová. Zatímco lineární strukturu vizualizujeme nejčastěji jako přímku a cyklickou jako kruh, stromová struktura je zobrazována jako graf s uzly a hranami, kde uzly jsou časová primitiva a hrany znázorňují časovou posloupnost mezi nimi. Více hran pak indikuje rozdělení časové linie. Lze tedy dobře znázornit alternativní scénáře pomocí větvení. Jelikož v naší problematice primárně chápeme čas jakožto sekvenční, budeme tedy volit časovou strukturu lineární, tedy čas budeme primárně vizualizovat za pomocí přímky, respektive osy.

Kritérium dat v zásadě představuje co konkrétně se vizualizuje ve vztahu

k časové ose, tedy primárně zahrnuje to, o čem jsme mluvili v kapitole zabývající se vlastními typy dat, viz. *Typy dat* 1.2.1.

Posledním kritériem je tedy reprezentace časově orientovaných dat. Zde opět můžeme rozlišovat dvě zásadní pod-kritéria: *časovou závislost* a *dimenzionalitu zobrazování*. Časovou závislost dělíme na statickou a dynamickou, přičemž první zmíněná forma zobrazuje data do statických, v čase neměnných obrazů. Oproti tomu dynamická časová závislost se snaží reprezentovat časovou závislost dat prostřednictvím reálného času, tedy prostřednictvím nějaké formy animace. Oba tyto přístupy mají své výhody a nevýhody; statické zobrazování například vyžaduje obrazovou reprezentaci času, např. prostřednictvím osy, čímž se zvyšuje množství vizuálního šumu. Výhodou pak může být vyobrazování všech informací současně do jednoho obrazu. Dynamická časová závislost poté může daleko lépe zobrazovat vývoj nějakého konkrétního datového bodu v čase, avšak animace jako takové představují vetší kognitivní náročnost pro uživatele, zejména v případě delších časových úseků s multivariantními daty. V našem případě se budeme omezovat pouze na závislost statickou, která bude vhodně doplněna o patřičnou formu uživatelské interakce, která bude do značné míry kompenzovat nedostatky tohoto přístupu.

Druhým podkritériem stran reprezentace je poté dimenzionalita zobrazování, což, jak již název napovídá, určuje, zdali prezentační prostor je ve 2D či 3D. V současné době se stále diskutuje nad použitím 3D vizualizace. Ve většině případů si totiž bohatě vystačíme s dvěma dimenzemi, přičemž použití 3D prostoru často představuje množství komplikací, které mohou vést i k horší orientaci či ke zkreslení dat. Z tohoto důvodu se tedy v našem případě budeme primárně soustředit právě na 2D zobrazování.

1.2.5.2 Vícenásobné zobrazení

Jednou ze zásadních metod při vizualizaci časově orientovaných dat (a nejen jich) je vícenásobné zobrazení, tedy vytvoření několika pohledů, které jsou v zásadě spolu propojeny, ale zároveň každý z nich může volit jinou vizualizační techniku a zobrazovat jinou část dat (roz. jiné proměnné/dimenze, vztahy). Toto řešení je reakcí na vícerozměrnost dat a především na avizovanou neexistenci jedné globální metody, jak časová data vizualizovat. Tyto pohledy se pak mohou zobrazovat paralelně, nebo pouze v omezené míře, v závislosti na tom, s jakými cíli v daném okamžiku data vizualizujeme. Obecně však platí že propojení několika různých zobrazení může vést k zefektivnění vizualizace jako takové. Při využití vícenásobného zobrazení je tedy primárním cílem k jednotlivým dílčím úkolům vizualizace vhodně zvolit příslušné techniky pro dané pohledy a tyto pak v případě nutnosti nějakým způsobem vhodně vizuálně propojit (např. brushing - vybraná podmnožina dat v jednom pohledu se zobrazí jako vybraná i v pohledu jiném). Vzhledem k tomu, že naše data jsou vícedimenzionální, není tedy příliš vhodné tyto zobrazovat

v jednom pohledu (zvláště když si uvědomíme podstatu vztahu server - VM je to téměř nemožné). Proto i v rámci našeho řešení budeme pracovat právě s několika pohledy, jmenovitě zvláště oddělíme celkové zatížení jednotlivých serverů a zatížení VM ke vztahu k jednomu serveru.

1.2.6 Uživatelská interakce

Uživatelská interakce zahrnuje všechny prostředky a procesy, při kterých uživatel na základě svého vstupu (nejčastěji prostřednictvím myší či klávesnice) organizuje, prozkoumává či jiným způsobem uspořádává vizualizaci jako takovou. Dobře zvolená interakce pak umožňuje uživateli lépe analyzovat a pochopit data nebo informace, která jsou mu vizualizací prezentována.

Je zřejmé, že díky obecně většímu rozsahu časově orientovaných dat je určitá forma interakce nezbytná pro jejich správnou analýzu. Tato interakce však nemusí být vždy v ohnisku zájmu, respektive mělo by být obecnou snahou dobrou formu interakce nabízet, avšak nevynucovat. Pokud podmíníme správné používání vizualizace tím, že zahltíme uživatele obrovským množstvím často nepotřebné funkcionality (zmáčknutí několika tlačítek naráz a zároveň pohyb myši na určité místo), může se tato stát zcela nepoužitelnou.

Rovněž je třeba příslušnou formu interaktivity volit v závislosti na konkrétních uživatelských cílech. Pokud je naším cílem monitorování a rychlá detekce anomalit (v našem případě právě stran vytíženosti serverů), je logické, že v takovémto případě je vhodné toto nepodmiňovat prakticky žádnou interakcí, která by tento proces zpomalovala. Naproti tomu pokud máme k dispozici velké množství dat, které reálně nelze vizualizovat naráz, je interakce nutná (vybrání konkrétního časového intervalu či zajímavé podmnožiny dat, pro které poté zkoumáme detail). V tomto směru je tedy nutné důkladně zvážit vazbu mezi zvolenou formou vizualizace a způsobem interakce s touto formou.

Další aspekt interakce, který je důležitý zejména v případě vícenásobného zobrazení jež budeme využívat, je tzv. koordinace, tj. šíření interakce (roz. výběru) vzniklé z jednoho pohledu na všechny ostatní pohledy, které jsou tímto způsobem "koordinovány". Zvláštní výzvou pak může být případ, kdy koordinujeme vizuální a analytické přístupy nesdílející stejné parametry. Příkladem takové situace může být kupříkladu koordinace dvou pohledů, z nichž jeden zobrazuje predikci budoucích trendů a druhý analýzu předchozího vývoje. V takovém případě je koordinace prakticky nemožná, respektive nemá smysl.

V neposlední řadě je také nutné zmínit vlastní formu interakce. V případě vizualizace, která je uskutečňována za pomocí výpočetní techniky, bývá obvyklé interakci zprostředkovávat primárně za pomocí myši a klávesnice, popřípadě dotykového displeje u mobilních zařízení. Alternativní variantou může potom být ovládání hlasem, které představuje mnohé výhody zejména ze strany pohodlnosti a u níž lze předpokládat dramatický rozvoj v rámci

příštích let. V našem případě se primárně omezíme na interakci za pomocí myši a klávesnice. Interakce pak bude vyžadována právě nejčastěji při výběru určitého časového intervalu (resp. okamžiku) a pro volbu mezi jednotlivými pohledy (zatížení serverů, zatížení VM, procesor vs. paměť).

1.2.7 Metody základní navigace v datech

Metody navigace ve vizualizovaných datech velmi úzce souvisejí s uživatelskou interakcí (de facto se jedná o specifické formy interakce jako takové), kdy uživatel jejím prostřednictvím mění rozložení a pohled na data s cílem lépe je analyzovat a získat tak cenné informace o nich. V následujících odstavcích popíšeme několik základních možností ve vztahu k vlastnímu průzkumu vizualizovaných dat.

1.2.7.1 Zooming

V některých případech je v rámci vizualizace a rozhraní jako takového potřeba, aby si uživatel mohl zvětšit příslušnou část pohledu za účelem lepšího přehledu při zkoumání detailů. Jelikož každý uživatel může chtít sledovat jinou část dat, respektive v průběhu času se tato oblast detailního pozorování mění, je potřeba aby přibližovací technika konzistentně škálovala nezávisle na této zóně. V případě uživatelského rozhraní, které zooming umožňuje, hovoříme v grafice o tzv. zoomable user interface, ve zkratce "ZUI". Dobře navržené ZUI tedy poté umožňuje uživateli měnit měřítko pohledu do různých úrovní detailu. Při zvětšování detailu pak hovoříme o zoom in technikách, při oddalování zoom out. Zoom se pak nejčastěji využívá při omezeném zobrazovacím prostoru, tedy zejména na mobilních zařízeních, nebo při velkém množství dat o malé granularitě. ZUI obecně umožňuje celkově produktivnější formu interakcí, díky kterým může uživatel lépe a rychleji data správně zanalyzovat.

1.2.7.2 Zoom and Pan

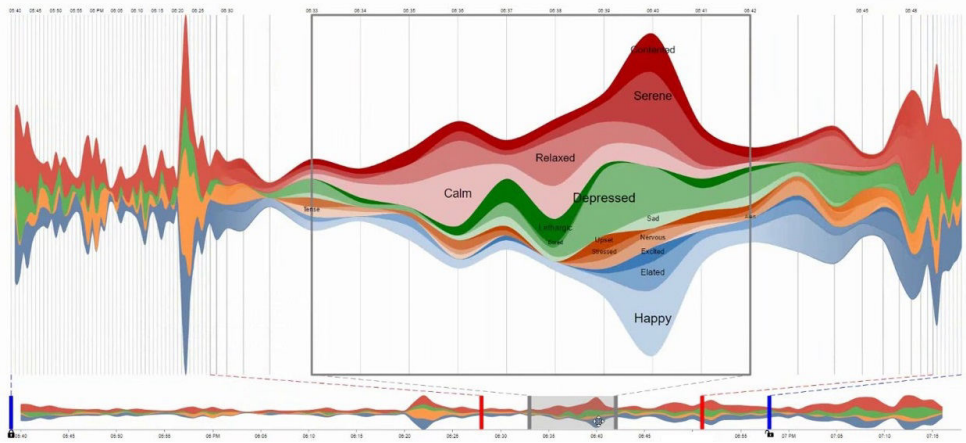
Tento přístup obohacuje ZUI o možnost posouvání rozsahu vizualizace v rámci zazoomovaného pohledu, a to aniž by bylo potřeba odzoomovat zpět do výchozí pozice a znovu přiblížit jinou její část. Toto přirozeně značně zrychluje manipulaci s vizualizací a v současné době tento přístup považujeme za standard. Hlavními výhodami této strategie je efektivní využití zobrazovacího prostoru a v zásadě neomezená škálovatelnost. Oproti tomu poměrně zásadní nevýhodou je výrazně zpomalená navigace vlivem absence globálního přehledu v rámci detailního pozorování, neboť detail zcela nahrazuje zobrazovací plochu výchozího přehledu.

1.2.7.3 Overview and Detail

Tento přístup počítá se zobrazováním více pohledů najednou, tedy s vícenásobným zobrazením 1.2.5.2. Jeden pohled reprezentuje obecný přehled nad celými daty, či jejich valnou většinou. Druhý poté vizualizuje vybranou detailnější část z této přehledové části. Tím je docíleno kýženého efektu, kdy je zachován přehled aby se zabránilo dezorientaci v detailu, zároveň pak kvůli oddělení těchto dvou pohledů přetrvává vizuální diskontinuita mezi nimi. Obecnou výhodou tedy je větší přehlednost a lepší navigace, nevýhodou poté větší prostorové nároky na vizualizaci (vícero pohledů). Tento přístup se v současnosti běžně aplikuje například v rámci map či prohlížení obrázkových kolekcí.

1.2.7.4 Focus and Context

Tato strategie rozšiřuje nebo zvětšuje oblast zaostření přímo v kontextu celkového pohledu nebo přehledů. Oblast uživatelského zájmu je zvětšena za účelem poskytnutí větší informační hustoty a lepší analýzu. Uživatel pak na základě posunu jednoduše změní tuto oblast zaostření, přičemž následně může detailněji analyzovat jinou část dat. Aby tento proces mohl fungovat v rámci jednoho pohledu (tedy zachování kontextu i detailu), je zapotřebí aby došlo k určité deformaci celé vizualizace, kdy měřítko přehledu je oproti měřítku detailu značně sníženo. Díky tomu je tato technika známá také jako deformované zobrazení. Základní předností je ponechání výhod přístupu Overview and Detail (tedy lepší navigace, znalost kontextu) s využitím daleko menšího zobrazovacího prostoru (jen jeden pohled), nevýhodou je pak již zmíněné zkreslení a deformace celkového přehledu, tedy kontextu.



Obrázek 1.5: Metoda Focus and Context v rámci proudového grafu [4]

1.2.7.5 Zvolený přístup

V rámci našeho řešení se jako nejlepší přístup v současné chvíli jeví určitá kombinace výše zmíněných přístupů. Předně je nezbytné využít přístup Overview and Detail, jehož využití je implicitně určeno typem našich dat. V rámci overview, jež bude primárně ohraničen zvoleným časovým úsekem, bude třeba zobrazovat celkové zatížení vlastních serverů (supervizorů), přičemž následně zobrazený detail bude představovat vizualizaci zatížení jednotlivých VM v rámci zvoleného serveru, opět pro určitý časový interval. Dále v případě detailu bude třeba umožnit uživateli lepší průzkum dat, a to primárně prostřednictvím Zoom and Pan, kdy půjde pomocí myši lépe manipulovat s danou vizualizací.

Kromě této základní konstrukce by bylo taktéž vhodné implementovat v rámci overview metodu Focus and Context, kdy ještě před tím, než uživatel definitivně zvolí časový úsek a server pro detailnější zobrazení, bude overview plynule reagovat na uživatelovu volbu a při již vybraných parametrech zobrazovat preview toho, co se v případě vizualizace detailu zobrazí. Toto preview pak bude zasazeno do kontextu celkového zatížení zvoleného serveru, což významně napomůže ke zrychlení orientace ve vizualizovaných datech.

1.2.8 Lidský faktor

Vizualizace pomáhá uživateli pochopit data za využití zraku jakožto silného paralelního vstupního kanálu vedoucího do lidské mysli. Pochopení omezení lidského vnímání, poznávání a jednání, stejně tak jako jejich celkový rozptyl, je pak nezbytné k tomu, abychom plně dokázali využít vizualizaci informací k podpoře rozhodování a k dosáhnutí vytyčených případů užití.

Je logické, že ve vztahu k vizualizaci nás především zajímají faktory, které nějakým způsobem ovlivňují zrakové vnímání člověka. Tyto faktory mohou

být dočasné či trvalé. Mezi dočasné většinou spadají stavy spojené s aktuálním rozpoložením daného člověka. Patří mezi ně kupříkladu netrpělivost, předsudky vůči systému, únava, rozptýlení či netrpělivost. Dočasné faktory musí být brány v potaz spíše v rámci testování; pokud kvalitu vizualizace testujeme na očividně unaveném člověku, nebo na někom kdo je nemocný, je logické, že výsledky nebudou nejlepší, což může vést ke zkresleným závěrům stran kvality řešení. Při vlastním návrhu vizualizace však na takovéto vlivy nehledíme (těžko budeme navrhovat speciální vizualizaci pro nemocného člověka, unaveného atd.), načež obecně dobrá vizualizace by neměla být těmito dočasnými faktory nějak drasticky znehodnocena.

Daleko závažnějším problémem je záležitost trvalých faktorů, respektive uživatelských indispozicí. Mezi tyto spadají zejména určitá poškození či nemoci vztahující se k očím, respektive k mozku uživatele. Nejčastějším případem je již dříve zmíněná barvoslepost, která je závažná zejména díky tomu, že je poměrně rozšířená (až 12% dospělých mužů má nějakou formu barvosleposti, u žen je pak tato porucha vzácnější). V případě barvoslepého uživatele je drastickým způsobem snížena možnost použití barvy jakožto rozlišovacího kanálu pro kategorická data. Rovněž obecné použití barvy může být v tomto případě značně omezené. Obdobným problémem může být v tomto směru i krátkozrakost, která obecně zhoršuje uživatelovu schopnost rozpoznávat malé detaily, respektive menší písmo/číslice. Z tohoto důvodu je potřeba volit dostatečně výraznou velikost písmen či číslic, případně umožnit uživateli si tuto velikost nastavit. Není pak asi nutné zmiňovat, že v případě úplné slepoty je vizualizace v tradičním slova smyslu pro takovou osobu zcela nepoužitelná.

Při navrhování našeho systému budeme mít tato možná zraková omezení na paměti, pravděpodobně se však kvůli podstatě našich dat nevyhneme tomu, že budeme nuceni využít barvený odstín, a to i v případě rozlišování kategorických dat (virtuální stroje, míra zatížení serverů). Do budoucna je tedy potřebná další diskuze k tomu, jakým způsobem výslednou vizualizaci upravit v případě, že uživatel bude postižen nějakou z výše zmíněných trvalých indispozic.

Kromě těchto zrakových indispozic lze mezi trvalé lidské faktory zahrnout i problémy s pamětí. Dobrá vizualizace se obecně snaží snížit co možná nejvíce potřebu si cokoliv pamatovat, avšak v některých případech se určité paměťové náročnosti nelze vyhnout. Rovněž rozdílnost krátkodobé a dlouhodobé paměti může hrát v určitých specializovaných případech problém (uživatelovi je předkládána informace, která je v rozporu s informací, jež má uloženou v dlouhodobé paměti - tendence později takovouto informaci podvědomě ignorovat). V našem případě však požadavky na paměť budou minimální, žádný problém by tedy v tomto směru u uživatele neměl vznikat.

Na závěr je také dobré zmínit pohybovou formu indispozice, která do značné míry může omezit či znemožnit uživatelskou interakci se systémem (nelze ovládat myš nebo klávesnici). Tuto možnost v našem řešení nepředpokládáme, avšak v současné době již existuje velká množina řešení, které

v takovémto případě uživatelskou interakci umožňují, a to nejčastěji za pomocí hlasové detekce.

1.2.9 Rizika vizualizace

Jako každý jiný vědní obor je i vizualizace spojena s určitými riziky, neboť vytvoření, respektive zvolení ideální metody pro splnění všech požadavků uživatelů představuje velkou výzvu. Jedním z nejčastějších problémů v rámci vizualizace je častá nutnost předchozí znalosti dat, případně vizualizační techniky. Vizualizace jako taková nemůže být vždy zcela intuitivní, zejména v rámci komplexnějších datových setů. V takových případech je nutné buďto předpokládat určitou znalost od uživatele, případně ho v průběhu vizualizace detailněji informovat prostřednictvím nápověd. To však značně zvyšuje kognitivní i perceptuální požadavky na použití dané vizualizace, což je často v přímém rozporu s její funkcí. Typicky pak platí, že data a zejména techniky, které nepotřebují žádné předchozí znalosti ze strany uživatele, jsou více obecné a nelze je tedy použít v plné míře při řešení specifických problémů.

Dalším významným problémem je škálovatelnost, čímž je především myšleno použití prostorově náročných vizualizačních technik v relativně limitované ploše, jakou může být například obrazovka mobilního telefonu. V takových situacích musí často docházet ke kompromisu, kdy se kvalita vizualizace sníží právě za cenu možnosti ji použít i pro menší obrazové plochy.

Rovněž estetická otázka je v současné době velmi diskutovaným aspektem vizualizace, neboť se ukázalo, že tato stránka zvolených technik má daleko větší význam z hlediska účinnosti vizualizace, než se původně předpokládalo. Estetika je navíc značně subjektivní oblastí, tedy je poměrně obtížné jí nějakým způsobem obecně vyhodnotit či kvantifikovat. Navíc od určitého množství dat je velmi náročné udržet určitou míru estetické přívětivosti, zvláště pokud nechceme snižovat informační hodnotu dané vizualizace.

Paradoxně jedním z velkých problémů může být vyhodnocování kvality dané vizualizace. Přestože existují některé základní předpoklady a poučky, podle kterých můžeme rychle objevit nedostatky daných řešení, ne vždy je tento proces zcela jednoznačný. Kupříkladu v současné době stále probíhá debata nad použitelností kruhového diagramu, jehož využití je v rámci vizualizace poměrně kontroverzním tématem. Navíc, jelikož vizualizace primárně cílí na usnadnění nějaké činnosti konkrétním lidem, je třeba vyhodnocování vztahovat konkrétně k těmto specifickým skupinám. Často se pak může ukázat, že určité řešení je ideální pro určitou část populace, zatímco zcela nepoužitelné pro část jinou, zejména v závislosti na věku.

V neposlední řadě je jedním z velkých rizik současné vizualizace určitá míra nejasnosti, a to ať už ze strany dat (ne vždy je jasné co v nich chceme objevit - "něco zajímavého"), či technik samotných (dokud techniku nenaimplementujeme a neotestujeme, není často jasné, jestli se pro danou věc vůbec hodí - znalost elementárních kognitivně-perceptuálních požadavků ne vždy stačí).

Jinými slovy, často nelze dopředu zvolit vhodnou vizualizační techniku jinak, než způsobem "pokus-omyl". Jelikož vytvoření jedné vizualizace může být poměrně časově i finančně náročné, není vždy žádoucí tento proces podstupovat vícekrát jen kvůli nalezení ideální vizualizace (jejíž kvalita bude stejně často předmětem sporu). Výsledek je tedy často suboptimální, byť i to může být dostačující (může být jedno jestli danou věc hledáme jednu sekundu nebo pět sekund, ale také nemusí).

1.3 Typické vizualizační techniky a jejich použitelnost

Se znalostmi s předchozí kapitoly se můžeme nyní, dříve než se podíváme na již existující řešení, nezaujatě podívat na potenciální vizualizační postupy a techniky a zhodnotit možnosti jejich využití ve vztahu k našemu problému, přičemž důraz tedy bude kladen na techniky, které v nějaké formě pracují s časem. Nejprve je však třeba si uvést několik základních chyb, kterých bychom se při obecném využívaní jakýchkoliv vizualizačních technik měli vyvarovat.

1.3.1 Nejčastější chyby při použití vizualizačních technik

V rámci návrhu a aplikace jednotlivých vizualizačních technik se často potkáváme s téměř až nepochopitelnými chybami, které jakožto uživatel dokážeme odhalit ihned. V rámci implementace těchto technik však můžeme často zdánlivě zjevné věci přehlédnout. Proto je potřeba při testovaní výsledného nástroje kontrolovat i aspekty, u kterých si chybu obecně nepřipouštíme.

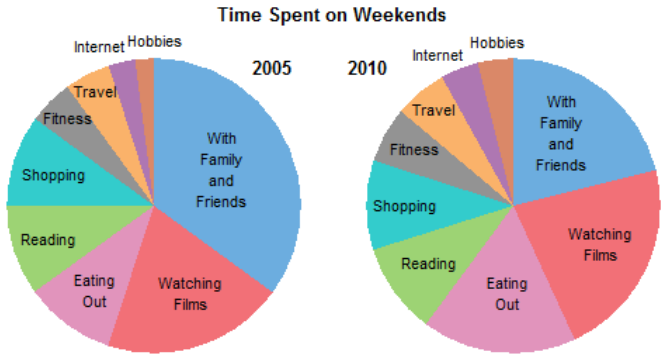
Jedna z nejnaivnějších chyb, na kterou můžeme v rámci vizualizace narazit, je chyba v rámci elementárních matematických operací. Tím je myšlen především fakt, kdy jednotlivé vizuální aspekty dramaticky neodpovídají doprovodným číselným vyjádřením (větší část kruhu představuje menší hodnotu), a zejména pak situace, kdy součet dílčích hodnot přesahuje maximální hranici. K této chybě může například dojít v případě, že vizualizované proměnné nejsou striktně disjunktní (mohou se prolínat) a my pro tuto skutečnost volíme nevhodnou techniku, kupříkladu kruhový diagram, složený sloupcový graf či složený plošný graf.

Dalším velkým problémem je nedodržování zavedených konvencí vizualizace. Experimentování v rámci aplikace zavedené techniky se ne vždy vyplácí, byť v některých případech může být přínosem. Typickým příkladem je pak nedodržení rozdělení os v grafu, kupříkladu jejich obrácení, kdy nula je nahoře (u osy Y, vpravo u osy X) a maximum dole (u osy X vlevo). Tato situace pak může vést k tomu, že výsledek analýzy dat je zcela opačný (v případě, že si uživatel nevšimne změny). Rovněž časovou osu je vhodné reprezentovat prostřednictvím osy X. Chybou je pak i oříznutí os, respektive nerovnoměrné měřítko os ve snaze ukázat větší objem dat naráz.

Jedním z problematických přístupů, který nemůžeme však obecně označit za chybu, je absence anotace. V častých případech je samotná vizualizace, která je doprovozena pouze o názvy os a její hodnoty, nedostačující, a tak musíme do obrazu přidat i kvalifikovaný text, který vizualizaci dodá smysl. Obecně by však vizualizace měla být vytvářena tak, aby míra nutné anotace byla minimální, respektive aby tato byla skryta a uživatel si jí mohl zobrazit pouze v případě, že jí potřebuje.

Zásadní problémem, který souvisí především s nevhodnou volbou vizualizační techniky jako takové, je nepatřičná vizualizace části dat. Tím je myšleno především vizualizace pouze určité části dat, resp. proměnných, za účelem nalezení určitého vztahu, který však bez detailnějších znalostí ostatních parametrů nemůžeme korektně podložit. Tato vizualizace je pak zavádějící a může vést uživatele ke špatnému závěru. Známým příkladem mohou být volební mapy, kdy určitá část země (kraj či region) je zabarvena podle toho, jaký kandidát či strana zde získala většinu. To nám však neříká vůbec nic o tom, jaký počet lidí v dané oblasti tento subjekt volilo a tedy zdali celkový procentuální počet voličů odpovídá procentuálnímu počtu získaných krajů (samozřejmě neodpovídá, neboť tyto části mají různé počty voličů).

Poslední větší chybou, na kterou můžeme při volbě vizualizační techniky narazit, je nevhodná metoda pro porovnávání dílčích částí kategorických dat. Pokud máme části (roz. vizualizované kategorie), které kromě velikosti mění i polohy, kupříkladu v rámci kruhového diagramu, je pro uživatele velmi obtížné rozdíl těchto částí přesně opticky určit, tak jak můžeme vidět na obrázku níže:

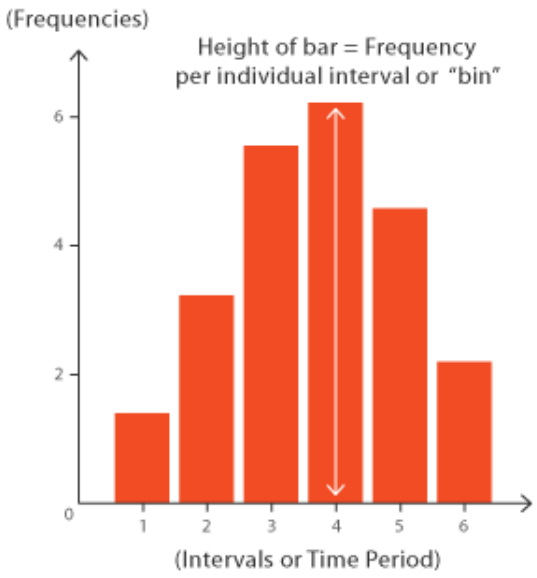


Obrázek 1.6: Z porovnávání obou kruhových diagramů nelze u některých výseků přesně určit, jakým způsobem se změnily (zvětšily/zmenšily), respektive o jakou hodnotu [19]

Tímto nedostatkem trpí velká část technik, jmenovitě treemapy, obecně složené grafy či již zmíněné kruhové diagramy a je proto vhodné tuto nedokonalost kompenzovat zcela jiným postupem, popřípadě předchozí techniku doplnit o další pohled, kde bude porovnávání řešeno intuitivnějším přístupem.

1.3.2 Histogram

Histogram je základní vizualizační technika časově orientovaných dat, jež byla poprvé použita Karlem Pearsonem v roce 1895. Zobrazuje distribuci dat v nepřetržitém časovém intervalu nebo v určitém období, jinými slovy osa X je typicky osou časovou. Každý sloupec v histogramu představuje četnost sledované veličiny (roz. konkrétní hodnoty) pro interval, respektive třídu (bin). Celková plocha histogramu se poté typicky rovná počtu nasbíraných dat. Histogramy poskytují odhad toho, kde jsou konkrétní hodnoty soustředěny, jaké jsou extrémy a zda jsou nějaké mezery mezi daty, popřípadě výskyty neobvyklých hodnot. Jsou také užitečné pro poskytnutí hrubého pohledu na rozdělení pravděpodobnosti.

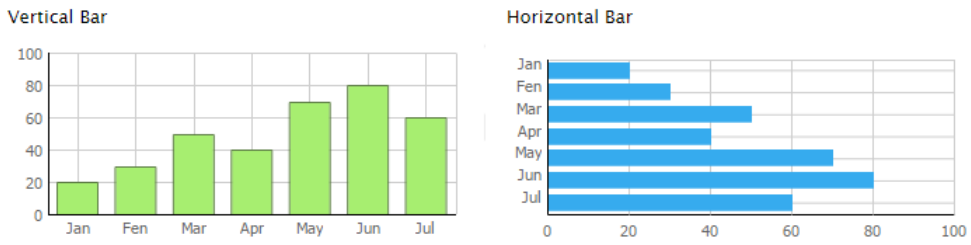


Obrázek 1.7: Histogram [20]

Z hlediska našeho zadání by bylo možné histogram využít v rámci zobrazování vývoje zatížení v čase, ať už jednoho serveru, nebo konkrétního virtuálního stroje. Tento singulární use case je však obecně spíše okrajový a v případě, že chceme zobrazovat současně celý vývoj pro daný server (tzn. všechny jeho VM a jejich vytížení v čase), je histogram nedostačující. Rovněž vývoj v čase, byť granularitou rozdělen na diskrétní, je v myslích uživatele vnímán spojitě a některé časové tendence mohou být tedy při přílišné diskretizaci histogramu nezřetelné.

1.3.3 Sloupcový graf

Sloupcový graf je na první pohled velmi podobný histogramu. Základní rozdíl spočívá v datech, se kterými sloupcový graf pracuje. Na rozdíl od histogramu, který zobrazuje časová data v rámci nějakého období, sloupcový graf typicky vizualizuje data kategorická, tedy umožňuje srovnání mezi jednotlivými kategoriemi. Tedy povětšinou jsou takto vizualizována diskrétní data, nikoliv kontinuální. Ta jsou poté zobrazována obdélníkovými sloupci s výškou nebo délkou úměrnou hodnotám, které reprezentují. Konkrétně tedy jedna osa grafu zobrazuje porovnávané kategorie a druhá osa představuje naměřenou hodnotu. Tyto sloupce mohou být zakresleny jak vertikálně, tak horizontálně (v případě histogramu v zásadě pouze vertikálně). Některé sloupcové grafy zobrazují sloupce seskupené ve skupinách, což umožňuje zobrazit hodnoty více než jedné měřené veličiny (viz. skládaný sloupcový graf níže).



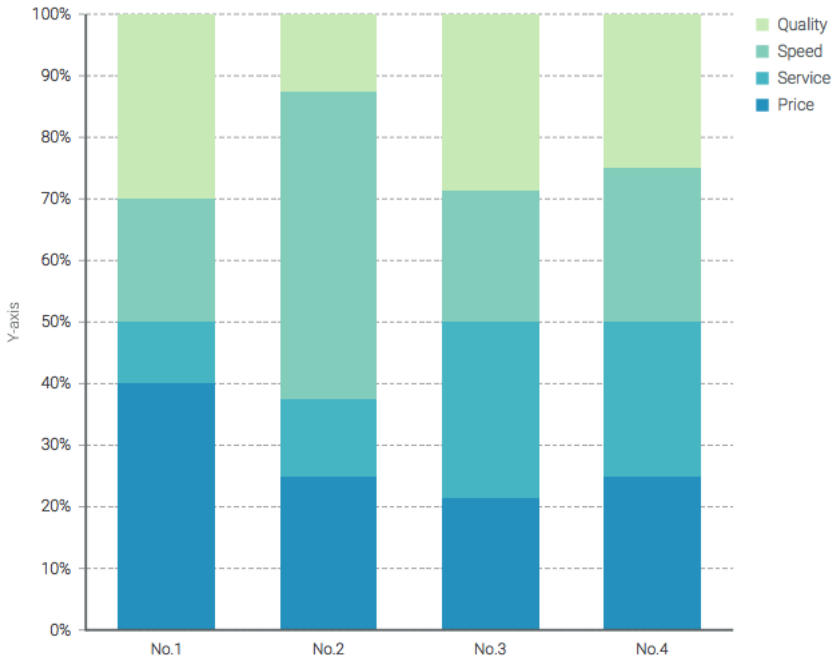
Obrázek 1.8: Vertikální a horizontální sloupcový graf [23]

Při využití sloupcového grafu (ať už horizontálního či vertikálního) v rámci našeho řešení narážíme na opačný problém než u histogramu; můžeme sice jednoduše zobrazit jednotlivé kategorie, tj. každý VM nebo server může být přítomen v rámci jednoho grafu, ovšem absence časové osy zapříčiní statičnost této vizualizace. Vývoj v čase by pak bylo nutné vizualizovat interakcí při přepínání časových okamžiků, což však klade zbytečně velké nároky na paměť uživatele, nehledě na to, že vypozorování trendů přes delší časový interval je v takovém případě velmi náročné, až nemožné.

1.3.4 Složený sloupcový graf

Složený sloupcový graf, jak již název napovídá, je jednou z možných variant klasického sloupcového grafu. Umožňuje v rámci jedné kategorie sledovat hodnoty více veličin, a to tak, že sloupec je rozdělen na více dílů, které jsou typicky barevně rozlišeny. Konkrétní barva pak může představovat hodnotu v rámci určité veličiny. Výška výsledného sloupce poté tedy zobrazuje kombinovaný výsledek jednotlivých veličin. Složené sloupcové grafy však obecně

nejsou vhodné pro datové sady, kde některé veličiny mohou mít záporné hodnoty (což však není náš případ).



Obrázek 1.9: Složený sloupcový graf [24]

Složený sloupcový graf trpí stále stejným problémem jako klasický sloupcový graf, tedy obtížností zobrazování více kategorií v rámci dlouhodobého časového vývoje. To lze do značné míry vyřešit překlopením kategorií a hodnot veličin; zatímco osa X se stane osou časovou, jednotlivé skupiny v rámci každého sloupce se mohou stát kategoriemi (roz. kupříkladu VM v našem případě). Přesto však i v takovémto případě je díky rozdělení sloupců a změny poměrů v rámci času relativně náročné rozeznat určité trendy a změny, respektive tato činnost vyžaduje stále poměrně velké kognitivní úsilí ze strany uživatele. V případě, že bychom však nechtěli zobrazit časový vývoj ale zajímal by nás naopak konkrétní časový okamžik (detail), by složený sloupcový graf mohl být poměrně vhodným kandidátem, zvláště díky malému obrazovému prostoru, který by v takovém případě vyžadoval (pouze jeden složený sloupec).

1.3.5 Svíčkový graf

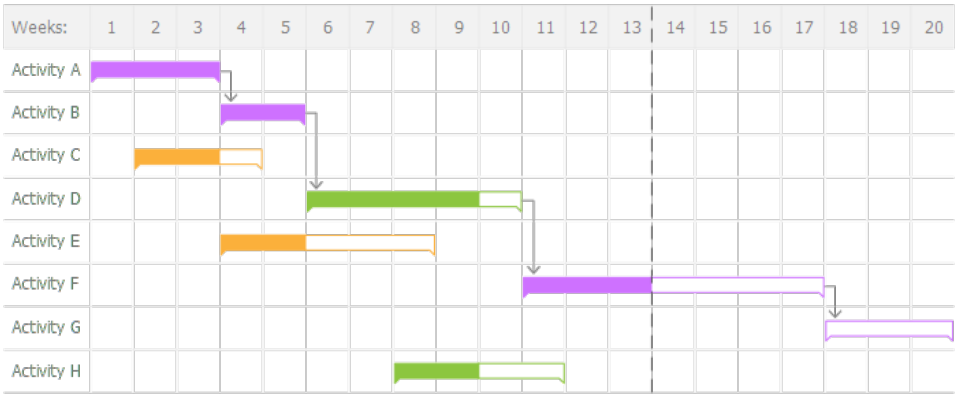
Svíčkový graf je poměrně stará vizualizační technika, která vznikla již kolem roku 1700 v Japonsku. Tento typ grafu je používán zejména jakožto jeden z nástrojů datové analýzy v rámci obchodování s komoditami a cenových pohybů v průběhu času, zejména pak měn či cenných papírů. Svíčkový graf se skládá z částí, které obecně nazýváme svíčky. Jedná se o sloupce (tzv. reálné tělo), jež jsou vertikálně protnuty úsečkami (tzv. stíny). Tyto sloupce pak nejsou vzhledem k ose Y pevně ukotveny (nezačínají v 0), tak jako v případě sloupcových grafů. Zatímco reálně tělo těchto svíček znázorňuje rozdíl mezi počáteční a koncovou hodnotou pro daný časový bod (resp. delší časový úsek znázorněný bodem na ose X - např. jeden den), stíny vyjadřují maximální a minimální hodnoty, kterých cena (či jiná proměnná) pro tento čas nabyla. Barva sloupců pak rychle indikuje, zdali cena celkově vzrostla, či klesla.

Obrázek 1.10: Svíčkový graf [20]

Zásadní problém svíčkového grafu je jeho prvotní menší intuitivnost. Uživatel tedy musí zprvu pochopit, jak tato technika funguje, což může zprvu zhoršit orientaci v daném problému. Rovněž, jak bylo řečeno na začátku, tato forma vizualizace je téměř vždy exkluzivně spjata s burzovním obchodem, její uplatnění v jiných oblastech je tak značně kontroverzní. Z tohoto důvodu i v našem případě nebudeme tuto možnost vizualizace uvažovat, přestože její určitá část by se v pravdě dala na náš problém aplikovat (např. maximální a minimální zatíženost VM v rámci intervalu, který je znázorněn jedním časovým bodem na ose X). Rovněž intuitivní pocit kontinuity času je v rámci svíčkového grafu díky diskretizaci jednotlivých sloupců značně omezen, což může být v našem případě nežádoucím účinkem.

1.3.6 Ganttův diagram

Ganttův diagram je vizualizační technika obvykle používaná jakožto organizační nástroj v rámci řízení projektů. Tyto typy grafů zobrazují seznamy aktivit (nebo úkolů) s jejich délkou v průběhu času, jež přesně znázorňuje, kdy každá aktivita začíná a končí. Díky tomu jsou Ganttovy diagramy velmi užitečné pro plánování a odhadování, jak dlouho může daná činnost (projekt) trvat. Rovněž umožňují zjistit, které aktivity probíhají mezi sebou paralelně. Ganttův diagram bývá zakreslen v tabulce; pro aktivity se pak používají řádky a sloupce reprezentují časová primitiva, tedy časový plán. Doba trvání každé aktivity je následně reprezentována délkou horizontálního sloupce, který pokrývá vyhrazené časové rozmezí. Dále lze použít barevné rozlišení sloupců k rozřazení jednotlivých činností do skupin. Pokud činnost právě probíhá, lze vizualizovat její procentuální dokončení odlišným barevným odstínem v části tohoto sloupce. Součástí diagramu jsou pak často i šipky, které určují návaznost jednotlivých aktivit na sebe či reprezentují kritickou cestu s klíčovými činnostmi. Rovněž významné milníky bývají znázorněny vertikálními čarami, které protínají celý graf v příslušném časovém bodě.



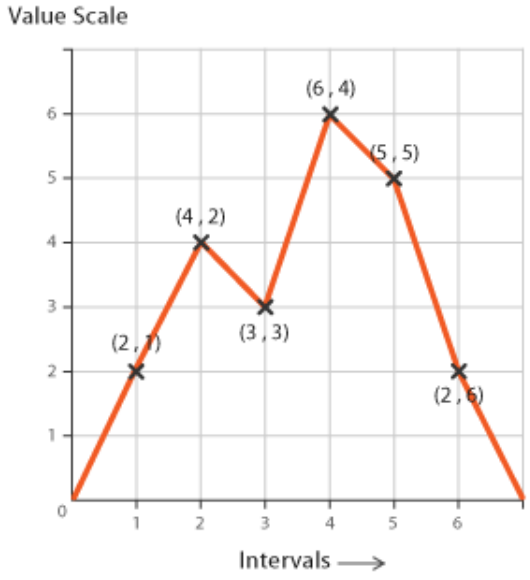
Obrázek 1.11: Ganttův diagram [20]

Už od pohledu je vidět zásadní problém v případě využití Ganttova diagramu v rámci naší problematiky, totiž že tato technika počítá s časově omezenými aktivitami, které jsou seřazeny do určitých sekvencí. Takovéto aktivity se v našem potenciálním řešení nevyskytují; zatížení může být sice spjato s určitým intervalem, avšak toto není podmíněno a ani nepodmiňuje zatížení předchozí či budoucí. Obecně se tedy dá říct, že žádné posloupnosti, které by bylo možné tímto způsobem vizualizovat, náš problém neobsahuje. Jelikož Ganttův diagram jako takový je pak navíc při větším množství aktivit horizontálně i vertikálně prostorově náročný, jeho využití pro případ vizualizace

několika desítek VM je kontraproduktivní.

1.3.7 Liniový graf

Liniový graf se používá k zobrazení kvantitativních hodnot v nepřetržitém intervalu nebo časovém období. Nejčastěji je využíván k zobrazení trendů a analýze toho, jak se data měnila v čase. Nejprve jsou vykresleny datové body na karteziánské souřadnicové mřížce a tyto jsou poté spojeny lomenou čárou (láme se v jednotlivých datových bodech). Typicky má osa Y kvantitativní hodnotu, zatímco osa X je časová osa nebo sekvence intervalů. Záporné hodnoty lze zobrazit pod osou X. Sklon linie, respektive jednotlivých segmentů, směrem vzhůru ukazuje, kde se hodnoty zvětšovaly, naopak sklon dolů kde hodnoty klesly. Cesta linie přes graf může vytvářet vzory, které odhalují trendy v datové sadě. Při zobrazování více datových skupin má typicky každá skupina svou vlastní linii, což umožňuje jednoduché porovnávání mezi nimi. Může to však zapříčinit i značnou nečitelnost grafu v případě, že obsahuje pět a více datových skupin, zvláště když se jednotlivé odpovídající segmenty protínají.



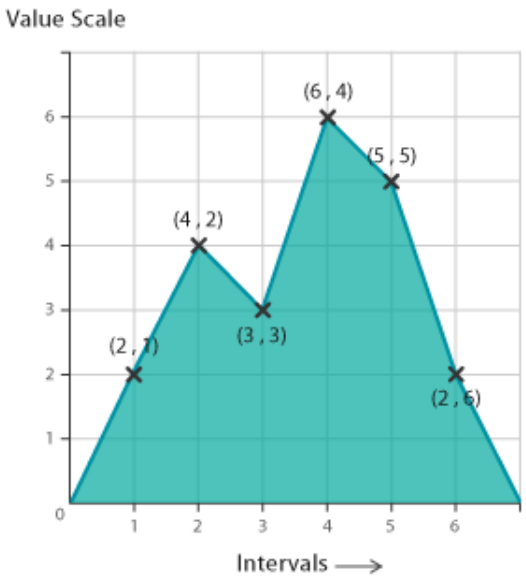
Obrázek 1.12: Liniový graf [20]

Hlavní problém při využití liniového grafu (známého též jako spojnicový graf či čárový graf) v rámci projektu je již zmíněná nečitelnost ve chvíli, kdy počet kategorií, které jsou jednotlivě mapovány na lomené čáry, překročí únosnou mez (10+). Vzhledem k tomu, že počty serverů a VM se pohybují v rámci

desítek, není tato technika pro zobrazení takovéhoto množství datových skupin vhodná. Jediné využití je tedy opět v případě limitovaného množství zobrazovaných kategorií (jednotky VM, porovnání dvou serveru apod.), což, jak již bylo zmíněno výše, není tak častý požadavek a je pro nás tedy vhodnější hledat v rámci naší problematiky flexibilnější vizualizační techniku.

1.3.8 Plošný graf

Plošné grafy jsou takové čárové grafy, kde je oblast pod lomenou čárou vyplněna určitou barvou nebo texturou. Plošné grafy jsou postupně vytvořeny nejprve vykreslením datových bodů na kartézskou souřadnicovou mřížku, dále spojením bodů lomenou čárou a nakonec vyplněním prostoru pod dokončenou čárou. Stejně jako liniové grafy se plošné grafy používají k zobrazení vývoje kvantitativních hodnot v intervalu nebo časovém období. Nejčastěji se používají k zobrazování trendů, spíše než ke sdělování konkrétních hodnot.

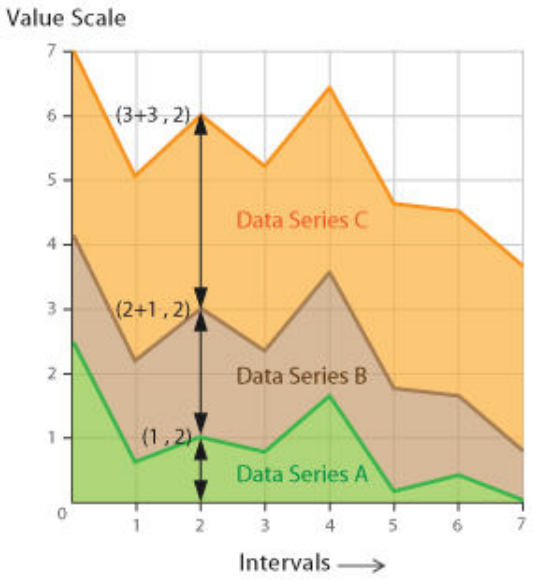


Obrázek 1.13: Plošný graf [20]

Z hlediska použití této vizualizační techniky trpí tradiční varianta plošného grafu stejným neduhem jako liniový graf - více datových sad je nečitelných, navíc postupné překrytí vybarvených ploch pod čarami může způsobit ještě horší orientaci, než v případě grafů čárových. Z tohoto důvodu se používají spíše varianty toho grafu, jakým je například složený plošný graf, viz. níže.

1.3.9 Složený plošný graf

Grafy složených ploch fungují obdobným způsobem jako základní varianta plošného grafu, s výjimkou použití více datových kategorií, které si v rámci grafu berou za počátek vždy bod, kde předchozí kategorie skončila. Celý graf pak představuje součet všech vykreslených dat postupně naskládaných na sebe, obdobně jako je tomu v případě složených sloupcových grafů. Složené plošné grafy rovněž používají v rámci kategorií výhradně kladná čísla, takže je nelze použít pro záporné hodnoty. Celkově jsou užitečné pro porovnání více proměnných nebo skupin, které se mění v rámci nějakého časového intervalu.



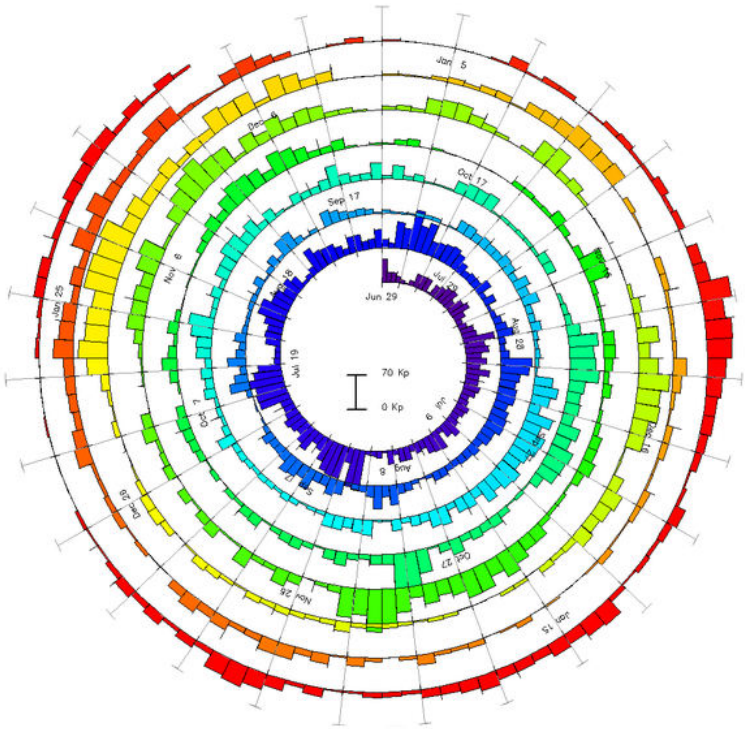
Obrázek 1.14: Složený plošný graf [20]

Využití složeného plošného grafu, na rozdíl od jeho klasické verze, může být v rámci našeho problému poměrně vhodné. Díky možnosti zobrazit více kategorií v rámci určeného časové intervalu (osa X jako časová osa), je možné jednotlivé virtuální stroje, popřípadě servery zobrazovat v tomto grafu jakožto spojité plochy. Můžeme tak vidět vývoj zatížení, pozorovat trendy a hledat maxima a minima, zvláště pokud je graf seřazen (nejvíce zatížený VM dole, nejméně nahoře). Problém může nastat, pokud je virtuálních strojů velké množství. Jakožto rozlišovací kanál mezi jednotlivými skupinami (VM) je zde totiž použita barva ploch, která ztrácí v případě velkého množství odstínů svou rozlišovací schopnost (existuje pouze omezený počet barevných odstínů, které zdravý člověk dokáže rychle a bezpečně rozeznat). To lze řešit zobrazením pouze omezeného množství VM/serverů, což lze v rámci monitoringu maxi-

málních/minimálních vytížení umožnit, v případě pozorování konkrétního VM v rámci kompletního kontextu však nikoliv. Rovněž při nízkých rozdílech vytížení a většího množství kategorií může být obtížné pro daný časový okamžik rozlišit jednotlivé poměry mezi konkrétními prvky. Z tohoto důvodu je v rámci detailu vhodné použít jinou, opticky více vypovídající vizualizační techniku, kupříkladu složený sloupcový graf.

1.3.10 Spirálový graf

Spirálový graf je poměrně atypický způsob vizualizace časově orientovaných dat, kdy tato data jsou mapována podél Archimedovské spirály, jež představuje časovou osu. Graf začíná ve středu spirály a postupuje směrem ven. Obecně pak platí, že vizualizační primitiva mohou být různých typů - linie, body, sloupce, plochy, případně i heatmapa aj. Lze tedy tvrdit, že spirálový graf představuje určitou variantu jiných, časově orientovaných vizualizačních technik. Velkou výhodou spirálového grafu je, že umožňuje poměrně dobře využívat zobrazovací prostor a díky tomu zvládá vykreslit poměrně velké datové sady, což může být přínosem při pozorování vývoje trendů v čase. Barva jednotlivých primitiv pak může kategorizovat jednotlivá období, která mezi sebou chceme porovnávat - kupříkladu jedna barva může znázorňovat jeden měsíc v roce, celý graf je pak vizualizací celého roku.

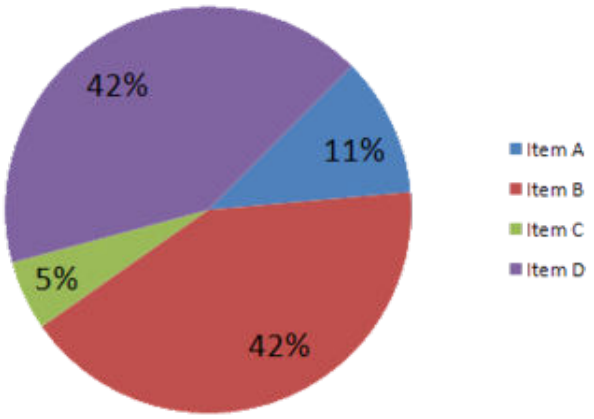


Obrázek 1.15: Spirálový graf [28]

Vzhledem k tomu, že v rámci našeho řešení budeme s největší pravděpodobností pracovat s většími datovými sadami, může řešení za využití spirálového grafu poskytovat určité výhody stran detailnějšího zobrazení většího časového rozsahu. Problémem však je, že díky zakřivení nemusí být jednoduché porovnávání několika hodnot zcela triviální, zvláště pak v případě, že jsou tyto více vzdáleny. Rovněž spirálový graf spíše počítá s vizualizací jedné proměnné v rámci času. Pokud bychom chtěli zobrazovat více hodnot (např. zatížení pro více VM), stal by se spirálový graf příliš vizuálně nečitelný. V případě zobrazení zatížení jednoho konkrétního serveru nebo virtuálního stroje přes delší časový úsek však spirálový graf může být vhodnou technikou.

1.3.11 Kruhový diagram

Kruhový diagram (známý také jako koláčový diagram nebo výsečový graf) je příklad kruhového statistického grafu, který je rozdělen na jednotlivé řezy ilustrující číselný poměr mezi dílčími kategoriemi a celkem, jež je tvořen celkovým součtem. V kruhovém diagramu je délka oblouku každého řezu (a následně tedy i jeho úhel a plocha) úměrná hodnotě, kterou daný řez reprezentuje. Nejčastěji se používají v případě vizualizace dat, které obecně můžeme považovat za části určitého celku, jehož velikost se nemění (roz. maximální hodnota). Kruhový diagram má poté velkou řadu variant, mezi které patří kupříkladu víceúrovňový kruhový diagram či prstencový graf.



Obrázek 1.16: Kruhový diagram [21]

Kruhové diagramy jsou dlouhodobě kritizovány mnohými odborníky za to, že je v nich obtížné porovnávat dílčí části bez dodatečného číselného vyjádření, popřípadě porovnávat data v rámci několika kruhových diagramů (tedy např. vývoj v čase). V rámci našeho projektu je možné kruhový diagram použít pro rozložení paměťového nebo procesorového zatížení konkrétního serveru

na jednotlivé VM, avšak absence časového aspektu neumožňuje dlouhodobé sledování trendů a rovněž velké množství dílčích částí může čitelnost diagramu významně omezit.

1.3.12 Graf Florence Nightingalové

Graf Florence Nightingalové (v odborné literatuře taktéž znám jako rose chart) je specifický graf, který lze považovat za určitou verzi kruhového diagramu nad opakující se časovou periodou. Byl poprvé použit anglickou ošetřovatelkou Florence Nightingalovou, která jeho prostřednictvím zobrazila počet úmrtí vojáků během krymské války (1853–1856). Tento graf používá polární souřadnicovou mřížku, jež je po obvodu rozdělena do několika částí. Tyto pak reprezentují nejčastěji periodicky se opakující časová primitiva, kupříkladu měsíc či týden. Dále jsou pak tyto části rozděleny, podobně jako složený sloupcový graf, do několika segmentů, jejichž vzdálenost od středu přímo závisí na hodnotě, kterou reprezentují. Každý z těchto prstenců tedy znázorňuje danou hodnotu pro konkrétní kruhovou výseč (časový interval) prostřednictvím svého obsahu. V praxi se tato vizualizační technika nejčastěji využívá v meteorologii.

Obrázek 1.17: Graf Florence Nightingalové [21]

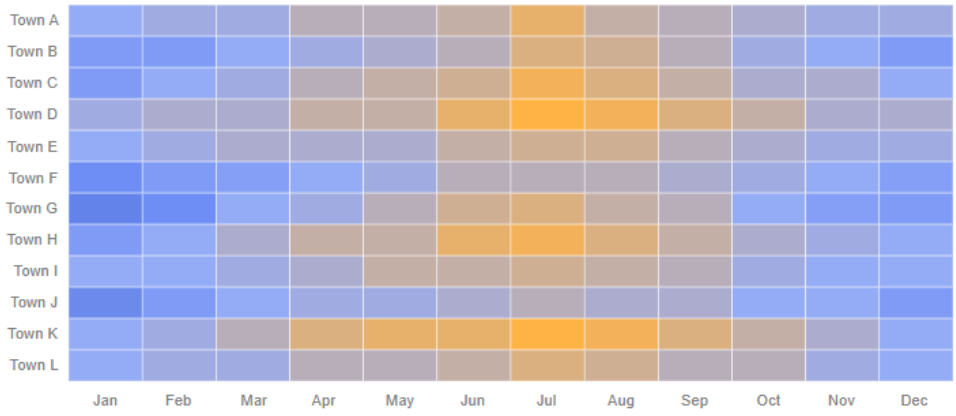
V prvé řadě je třeba říct, že ze všech zde popsaných technik je tento typ grafu nejméně intuitivní. To samo o sobě značí, že tuto metodu nebude příliš vhodné použít. Navíc její použití předpokládá určitou periodicitu času, s kterou v našem případě primárně nepracujeme. Další nevýhodou je fakt, že

vnější segmenty, které jsou dál od středu, zabírají výrazně větší plochu grafu, což typicky způsobuje nepřiměřené vnímání větších hodnot.

1.3.13 Heatmapa

Heatmapy vizualizují data prostřednictvím změny v barevném odstínu, popřípadě intenzity. Heatmapy jsou typicky členěny v rámci tabulkového formátu a umožňují tak zkoumat vícerozměrná data. Heatmapy jsou dobré pro zobrazování rozptylu mezi více proměnnými, odhalují vzory, zobrazují, zda jsou nějaké proměnné navzájem podobné a umožňují zjistit jestli mezi nimi existuje nějaká korelace.

Typicky jsou v rámci heatmap řádky tabulky objekty jedné kategorie, naproti tomu sloupce představují kategorii druhou. Jednotlivé řádky a sloupce se poté protínají v rámci buněk tabulky (resp. matice) a údaje v nich obsažené jsou tedy založeny na vztahu mezi dvěma proměnnými. Buňky pak buď obsahují barevně kódovaná kategorická data, nebo číselná data založená na barevné škále. Vzhledem k tomu, že heatmapy spoléhají na barevné odstíny jakožto kanál pro komunikaci informací uživateli, jsou tyto spíše vhodnější pro zobrazení obecnějšího pohledu na číselné údaje, neboť je těžší přesně vypozorovat rozdíly mezi barevnými odstíny a extrahovat tak konkrétní hodnoty. V neposlední řadě lze také tuto techniku použít k zobrazení změn dat v čase, a to pokud je jeden z řádků nebo sloupců nastaven na časové intervaly.



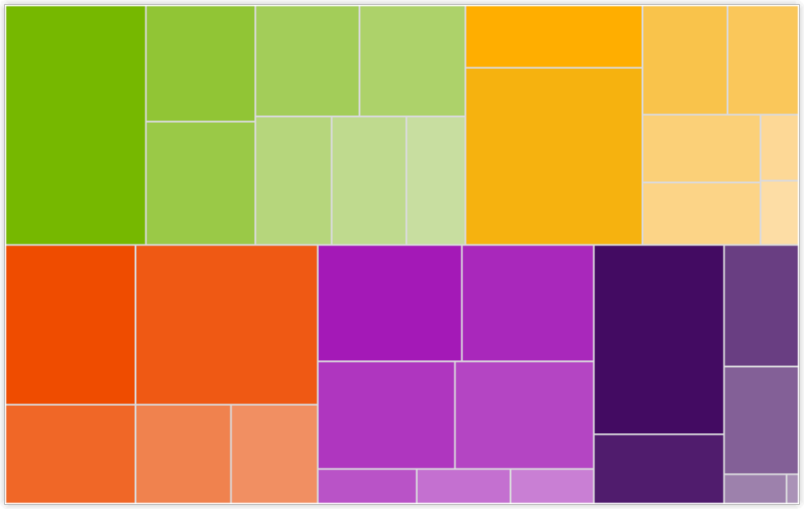
Obrázek 1.18: Heatmapa [20]

Heatmapy mohou v našem problému hrát roli orientační vizualizační techniky, která uživateli v krátkém čase zobrazí agregaci nějakých dat a rychle mu tak přiblíží celkovou situaci. Kupříkladu v rámci use case monitoringu chceme, aby uživatel co nejrychleji zaznamenal vzniklý problém (přílišné vytížení serveru). Za tímto účelem je možné využít heatmapu společně s vhodným barev-

ným schématem, kde výrazné změny ve vytížení jednotlivých serverů v rámci časových okamžiků bude možné rychle indikovat právě výrazným barevným rozdílem. Díky časové ose bude zároveň možné sledování obecných trendů při změnách vytížení.

1.3.14 Treemapa

Treemapa je vizualizační technika pro zobrazování hierarchických dat pomocí vnořených struktur, nejčastěji obdélníků. Každá větev stromu je obdélník, který je rekurzivně rozdělený do menších, jež představujícími dílčí větve. Obdélník listového uzlu má poté oblast úměrnou stanovenému rozměru dat. Jednotlivé obdélníky jsou pak často ještě barevně rozlišeny, což umožňuje mapování dalšího rozměru dat. V případě, že barvy a velikosti obdélníků jsou nějakým způsobem spojeny se strukturou stromu, lze často velmi rychle spatřit vzory, které by byly obtížně rozpoznatelné v rámci jiných vizualizačních technik. Druhou výhodou treemapy je, že při konstrukci efektivně využívá prostor. V důsledku toho může uživatel na obrazovce číst relativně čitelně tisíce položek současně.

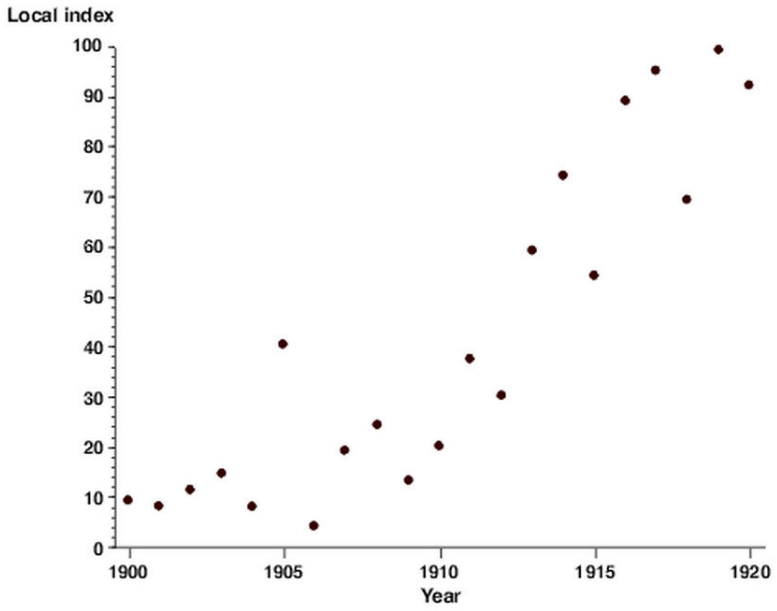


Obrázek 1.19: Treemapa [25]

Využití treemapy v rámci naší problematiky se nabízí v případě hierarchického vztahu mezi serverem a jeho virtuálními stroji. Tedy každý stroj by mohl být v konkrétní treemapě (jedna treemapa = jeden server) zobrazen jako jeden obdélník s obsahem, který by odpovídal poměru vytíženosti daného VM ve vztahu k serveru. Opět zde však absentuje časový údaj a tím pádem by treemapa mohla být využita pouze v rámci vyobrazení dat pro konkrétní časový interval.

1.3.15 Korelační diagram

Korelační diagram je typ grafu, respektive matematického diagramu, který v karteziánských souřadnicích zobrazuje typicky hodnoty dvou proměnných pro daný dataset. Data jsou zobrazena jako množina bodů (teček), z nichž každý má hodnotu jedné proměnné určující polohu na vodorovné ose a hodnotu další proměnné určující polohu na svislé ose. Pokud budeme jednotlivé body odlišovat barvou, můžeme jejím prostřednictvím zobrazovat další dimenzi dat.



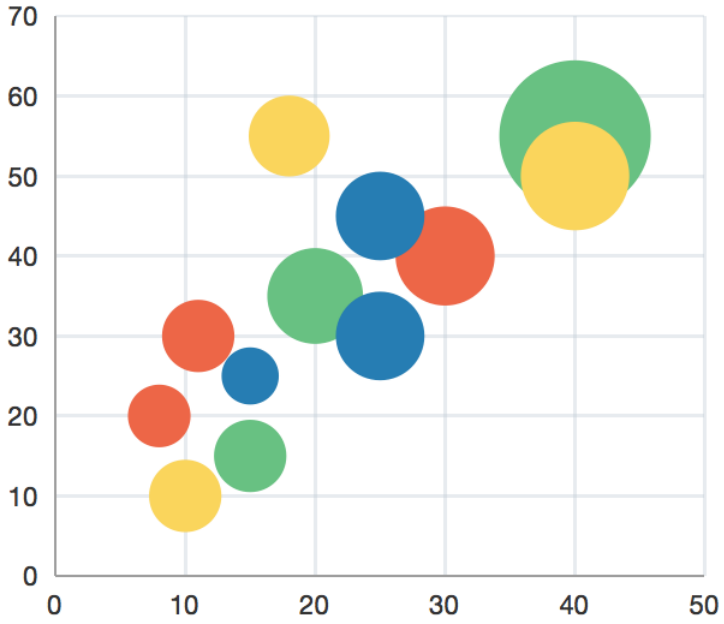
Obrázek 1.20: Korelační diagram [26]

V rámci využití korelačního diagramu pro naše řešení se jakožto největší potenciální problém ukazuje velké množství datových záznamů a velká diskretizace vizualizovaných bodů. To může způsobit přemíru vizuálních informací a významně zhoršit orientaci při řešení konkrétního use casu, zejména pro monitoring trendů v rámci velkého množství VM v delším časovém období.

1.3.16 Bublinový graf

Bublinový graf je multivariantní graf, který lze v pravdě chápat jako sjednocení korelačního diagramu se specifickou formou plošného grafu. Stejně jako korelační diagram, i bublinový graf používá karteziánský souřadný systém k vykreslování bodů v rámci mřížky, kde osy X a Y reprezentují oddělené proměnné. Nicméně na rozdíl od diagramu jsou jednotlivé body v rámci bublinového grafu barevně rozlišeny k určení specifické kategorie či hodnoty pro-

měnné dalšího typu. Navíc, velikost těchto bodů (obsah kruhu) může reprezentovat čtvrtou nezávislou proměnnou. Čas je poté zobrazen prostřednictví jedné z os (typicky X), popřípadě animací těchto datových proměnných měnících se v průběhu času. Bublinové grafy se obvykle používají k porovnávání a zobrazování vztahů mezi kategorizovanými body (kruhy) za využití jejich umístění a proporcí. Celkový obraz bublinových grafů lze pak nejčastěji využít právě k analýze vzorů a korelací.



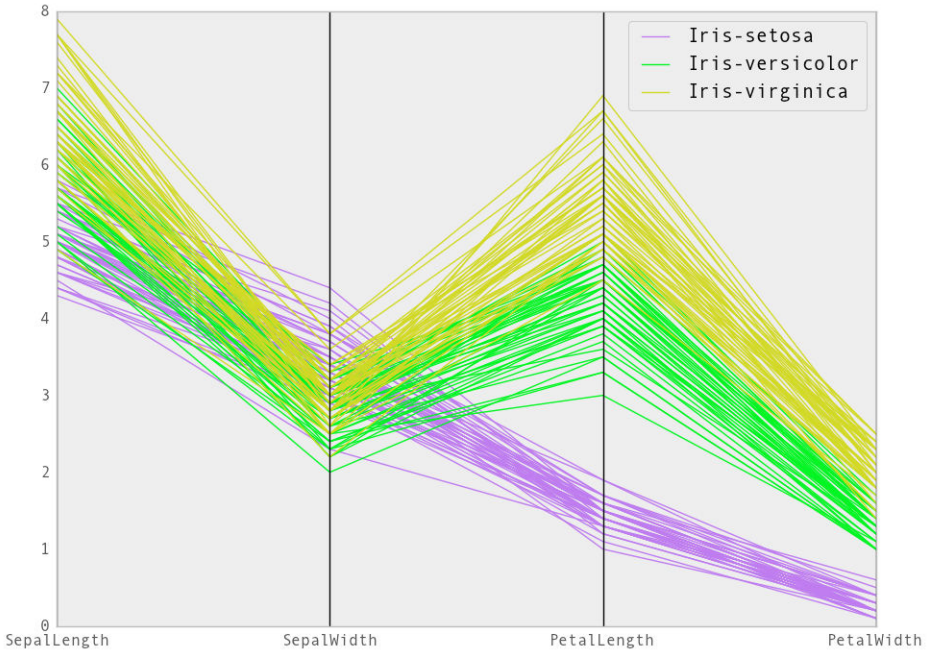
Obrázek 1.21: Bublinový graf [22]

Zásadním problémem je rapidně se snižující čitelnost grafu v případě velkého množství bublin, tedy jednotlivých datových objektů. Bublinové grafy tak mají značně omezenou kapacitu stran velikost vizualizovaných dat, byť tento problém se dá do určité míry řešit za pomocí interakce. Z důvodu velkého množství dat v případě naší problematiky nebude tedy tato vizualizační technika použita, byť v rámci kategorických dat (jedna bublina by reprezentovala konkrétní VM či server) má své uplatnění. Navíc hlavní výhoda bublinových grafů, tedy velké množství proměnných které můžeme vizualizovat naráz, by zůstala do značné míry nevyužita - v rámci naší vizualizace budeme uvažovat maximálně tři proměnné v rámci jednoho pohledu.

1.3.17 Rovnoběžné souřadnice

Paralelní souřadnice jsou typickým způsobem vizualizace n-dimenzionálních dat a jejich následné analýzy. Při této vizualizační technice jsou jednotlivé

dimenze dat mapovány na rovnoběžné horizontální či vertikální osy, z nichž každá reprezentuje škálu pro jednu konkrétní dimenzi. Bod v n-dimenzionálním prostoru je poté reprezentován jako lomená čára s vrcholy na příslušných rovnoběžných osách, které odpovídají i-té souřadnici daného bodu. Tato vizualizace vetšinou nepracuje s časově závislými daty, respektive vizualizuje data pro daný časový okamžik, popřípadě podmiňuje zobrazení časového intervalu výběrem.



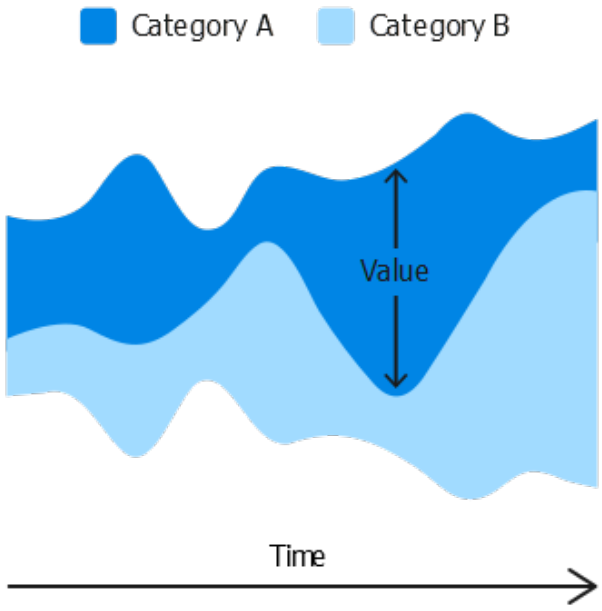
Obrázek 1.22: Rovnoběžné souřadnice [27]

V případě použití rovnoběžných souřadnicí v našem řešení narážíme na poměrně zásadní problém, a to stran viditelnosti sledovaných dat. Při typickém použití této vizualizační techniky je sledování konkrétních hodnot podmíněno výběrem, což může v případě monitoringu představovat problém (snažíme se v rámci monitoringu o co nejmenší vynucenou interakci ze strany uživatele). Rovněž horší rozpoznatelnost trendů v čase může hrát významnou roli při zvolení, respektive nezvolení této techniky pro naše řešení.

1.3.18 Proudový graf

Tento typ vizualizace je variací složeného plošného grafu, avšak namísto vykreslování hodnot proti pevné, horizontální ose zobrazuje proudový graf hodnoty kolem měnící se základní centrální linie. Vývoj hodnot různých kategorií v čase je poté zobrazován pomocí křivek, které se poněkud podobají proudu vody v řece (odtud tedy proudový graf). To může na první pohled činit graf

esteticky příjemným a více poutavým. V proudovém grafu je velikost každého jednotlivého tvaru toku úměrná hodnotám v každé kategorii. Osa, kolem které jednotlivé toky proudí paralelně, se používá jako osa časová. Barva může být použita buď pro rozlišení každé kategorie, nebo pro zobrazení dalších kvantitativních hodnot jednotlivých kategorií právě změnou barevného odstínu. Tato vizualizační technika je tedy ideální pro zobrazování svazků dat velkého objemu a objevování trendů a vzorů v průběhu času v rámci většího počtu kategorií. Proudové grafy lze také použít k vizualizaci volatility pro velkou skupinu aktiv během určitého časového období.



Obrázek 1.23: Proudový graf [20]

V případě možnosti využití proudového grafu, respektive rozdílu oproti použití složeného plošného grafu, můžeme narazit na zhoršené vnímání rozdílů a tedy i porovnávání jednotlivých kategorií navzájem, a to zejména při velkém počtu kategorií. To může být právě způsobeno faktem, že proudový graf nemá pevnou časovou osu, nad kterou by se graf zobrazoval. Jednotlivé křivky se tedy nachází jak nad, tak i pod touto osou. Obecně je však využití proudového grafu v zásadě validní možností jakožto jedna z možných variant složeného plošného grafu.

1.3.19 Shrnutí vizualizačních technik

Nyní můžeme provést prvotní shrnutí zmíněných vizualizačních technik ve vztahu k použitelnosti v rámci naší problematiky:

- **Histogram**
  Histogram jako takový trpí zejména omezenou možností zobrazování vícera dimenzí a taktéž diskretizací časových okamžiků, díky čemuž může být zhoršena schopnost pozorovat vývoje trendů. Rovněž kategorizace dat může v rámci přehlednosti histogramu způsobovat značné problémy.

- **Sloupcový graf**
  Sloupcový graf se prioritně soustřeďuje na kategorická data, dobře tedy může vyobrazovat hodnoty konkrétních serverů nebo VM. Daleko hůře však v takovémto případě zobrazuje časový vývoj, a to právě díky absenci časové osy; celkový pohled je statický a změna časového intervalu je tedy podmíněna interakcí, což pro porovnávání více časových okamžiků není vhodné.

- **Složený sloupcový graf**
  Do značné míry řeší problém sloupcového grafu, a to přidáním další dimenze prostřednictvím rozdělení sloupců do více kategorií. Oproti jiným vizualizacím je časový vývoj však opticky diskretizován a významné rozdíly mezi sousedními intervaly (absence několika VM, přidání nových) mohou zapříčinit zhoršenou orientaci při sledování vývoje trendů. Je však vhodným kandidátem pro zobrazení poměru jednotlivých kategorií pro konkrétní časový okamžik, rovněž také využívá poměrně malého prostoru - dobré pro detailní pohled.

- **Svíčkový graf**
  Tato technika je primárně určena pro obchodní vizualizaci, navíc prakticky nedisponuje možností využít barvu pro kategorizaci dat (jednotlivé virtuální stroje); barva je zde vázána k rozlišování růstu či poklesu hodnoty pro dané časové primitivum. Tím se značně snižuje možná dimenzionalita vizualizovaných dat. Rovněž kontinuálnost času pro pozorování trendů je díky viditelné diskretizaci časových bodů značně omezena, obdobně jako u histogramu.

- **Ganttův diagram**
  Jelikož řešení našeho problému nebude s největší pravěpodobností pracovat s žádnými časovými posloupnostmi (vytížení se vizualizuje konstantně) ani aktivitami, které mají vyhrazený určitý časový interval, je použití Ganntova diagramu nevhodné, zvláště pak když vezmeme v úvahu jeho problematickou čitelnost při větším objemu vizualizovaných dat.

- **Liniový graf**
  Hlavním problémem při použití liniového grafu, který jinak dobře zobrazuje vývoj trendů v čase, nastává v případě velkého množství sledovaných objektů; mnoho navzájem protínajících se lomených čar (10+) rapidně zhoršuje čitelnost grafu a vyžaduje selekci od uživatele pro zpřehlednění.

- **Plošný graf**
  Plošný graf má v zásadě stejné problémy jako jsou v případě liniového grafu; dobré zobrazování trendů je podmíněno malým množstvím průniků jednotlivých ploch, ke kterým navíc dochází v případě plošného grafu vždy, i při menším počtu sledovaných objektů. Pro velké množství kategorií neumí dobře zobrazit poměry mezi nimi, a to právě díky vzájemnému překryvu. Díky desítkám různých virtuálních strojů v rámci jednoho serveru není tato technika pro naše řešení vhodná.

- **Složený plošný graf**
  Nejlepší kandidát pro vývoj zatížení virtuálních strojů v čase. Překryv ploch je zde eliminován posunutím, díky interpolaci jsou plochy spojité a lze tak dobře sledovat změny v daném intervalu. Horší může být porovnávání jednoho či několika diskrétních okamžiků, a to zejména díky optickému zkreslení proměnlivým spojitým okolím. Za tímto účelem je vhodné pro tento případ použít detailní pohled soustředící se na konkrétní okamžiky - lepší technika pro zobrazení poměrů, například složený sloupcový graf. Rovněž barevné rozlišení většího počtu virtuálních strojů může být problematické.

- **Spirálový graf**
  Spirálový graf může být do značné míry vhodným řešením při zobrazování většího množství dat, respektive při detailnějším pohledu na delší časový interval. Naproti tomu je však poměrně nevhodný pro porovnávání konkrétních hodnot, rovněž při větší dimenzionalitě dat je čitelnost grafu značně snížena. Z tohoto důvodu v rámci našeho řešení nebudeme spirálový graf primárně používat, zůstává však jednou z lepší variant a v případě malého zobrazovacího prostoru může být do značné míry jedním z mála vhodných řešení stran maximálního využití tohoto prostoru.

- **Kruhový diagram**
  Může být poměrně dobrý pro orientační vhled do celkového zatížení v rámci konkrétního okamžiku, v případě většího množství dílčích částí však ztrácí na přehlednosti. Rovněž porovnávání několika kruhových diagramů mezi sebou (například každý pro různý časový okamžik) je kognitivně náročné. Lepší alternativou proto může být v případě většího množství sledovaných objektů složený sloupcový graf.

- **Graf Florence Nightingalové**
  Tato technika trpí na poměrně malou čitelnost a značnou míru zkreslení, což je jak v případě rychlé detekce anomálií, tak při porovnávání několika hodnot s cílem pozorování trendů poměrně zásadním problémem. Navíc vyžadovaná periodicita času značně omezuje možnosti vizualizace, kdy je třeba omezovat se na opakující se intervaly. Z těchto důvodu není tato metoda v rámci naší problematiky vhodným přístupem.

- **Heatmapa**
  Nejvhodnější vizualizační technika pro use case monitoringu, respektive pro rychlou detekci abnormalit. Díky barevnému pop up efektu umožní bez větších obtíží velmi rychle indikovat vznikající problém či abnormality. Jako taková je však vhodnější pro celkový overview, než pro detailnější zkoumání, neboť vztah mezi číselnou hodnotou a barevným odstínem není vizuálně přesný.

- **Treemapa**
  Treemapa je poměrně vhodná pro zobrazení většího množství kategorických dat a poměrů mezi jejich hodnotami, což v rámci naší problematiky může být případ vizualizace rozložení zatížení hostovacího serveru na zatížení dílčích VM. Hodnoty, které si jsou však poměrně blízké jsou v rámci treemapy hůře rozlišitelné, rovněž velké rozdíly mezi vytíženími zapříčiní, že některé virtuální stroje budou namapovány na velmi malé obdélníky. Treemapa rovněž zobrazuje jen konkrétní časový interval či okamžik, nedají se tedy v jejím případě pozorovat změny v čase.

- **Korelační diagram**
  Korelační diagram v našem případě není vhodnou vizualizační technikou, neboť je při velkém množstvím datových záznamů (což je náš případ) téměř nemožné vypozorovat vývoj pro konkrétní objekty, zvláště díky velké diskretizaci a potenciálnímu zahlcení vyhrazené plochy datovými body.

- **Bublinový graf**
  Z obdobného důvodu jako korelační diagram není ani bublinový graf vhodnou vizualizační technikou pro řešení naší problematiky. Jeho nízká čitelnost při velkém objemu dat je navíc oproti korelačnímu diagramu ještě umocněna faktem, že jednotlivé bubliny zabírají poměrně hodně zobrazovacího prostoru.

- **Rovnoběžné souřadnice**
  Rovnoběžné souřadnice jako takové umožňují poměrně dobré zobrazení vícedimenzionálních dat, hlavním problémem je však zobrazení vývoje v čase, který je obecně v rámci této vizualizační techniky problematický. Orientace v grafu při velkém množství dat je rovněž podmíněna selekcí.

To už samo o sobě není v našem případě vhodné, neboť chceme, aby uživatel viděl souvislosti pokud možno ihned.

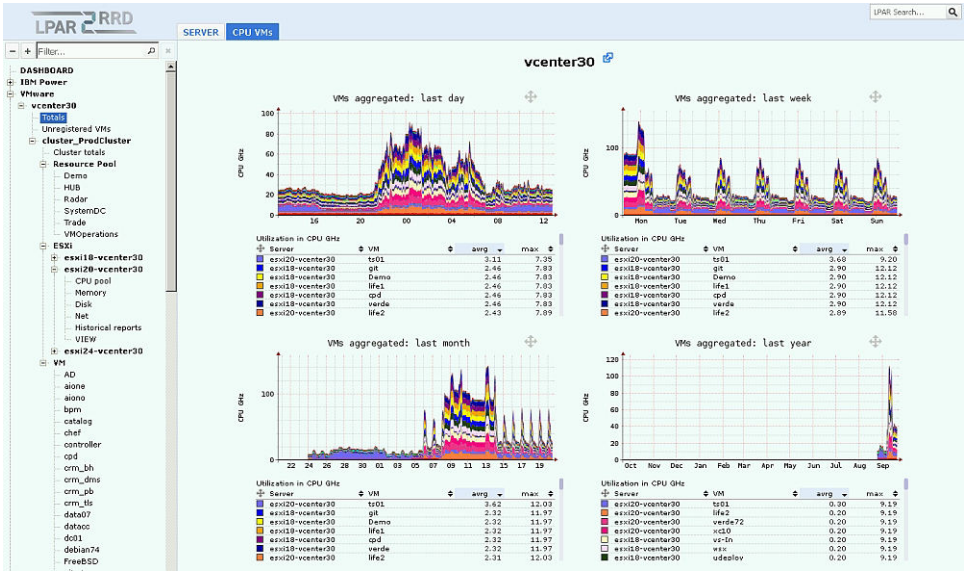
- **Proudový graf**
  Jedna z možných alternativ k složenému plošnému grafu, u které je však největším problémem jakási vizuální nestabilita způsobena nepevnou středovou osou. Navíc část pod touto osou nemá v rámci naší problematiky využití, respektive je redundantní a jejím zobrazením bychom se připravovali o zobrazovací prostor, obecně pak zhoršuje vnímání poměrů mezi jednotlivými částmi.

1.4 Analýza existujících řešení

Nyní, poté co jsme se obecně zabývali jednotlivými nejběžnějšími technikami, se můžeme detailněji podívat na to, jakým způsobem se problematika zatížení serverových cloudů, respektive vizualizace tohoto zatížení, řeší v praxi. Ve většině případů je vizualizace paměťového a procesorového zatížení součástí rozsáhlejších nástrojů, které umožňují, mimo jiné, měnit vizualizační techniky právě za účelem lepší orientace a sledování různých aspektů stavu cloudu.

1.4.1 lpar2rrd

Lpar2rrd představuje zdarma dostupný nástroj pro monitorování výkonu systémů jako je VMware ™ a IBM Power Systems ™. Nástroj shromažďuje data o výkonu a umožňuje pak zobrazit současné, minulé a budoucí trendy za pomoci grafů mapujících virtuální prostředí. Je rovněž agent-less, veškerá požadovaná data jsou tedy stažena z řídících stanic (vCenter nebo HMC).

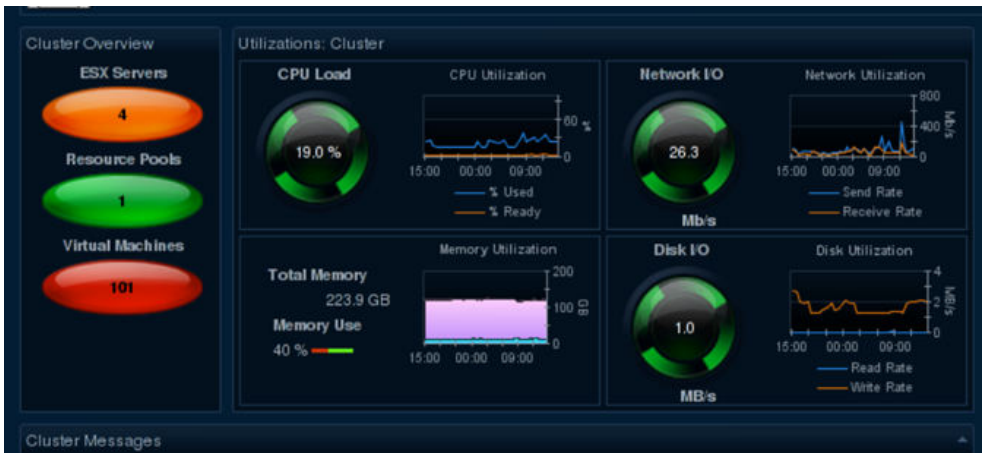


Obrázek 1.24: Dominující použití složeného plošného grafu v rámci nástroje Lpar2rrd [35]

Jak lze z obrázků vidět, prioritní technikou vizualizace procesorového a paměťového zatížení ve vztahu server-VM je v případě Lpar2rrd plošný graf, respektive složený plošný graf. Ten poměrně dobře zobrazuje vývoj zatížení v čase a umožňuje tak v rámci monitoringu pružně a rychle reagovat na případné změny a problémy.

1.4.2 vFoglight

VFoglight (dříve Vizioncore vFoglight) společnosti Quest Software Inc. je jedním z mála multiplatformních nástrojů pro měření výkonu. Dokáže monitorovat jak VMware vSphere, tak i Microsoft Hyper-V. VFoglight má jedno z nejhezčích rozhraní, ale může vyžadovat značné množství zdrojů a celkově tak zatěžovat vlastní systém, na kterém nástroj běží.

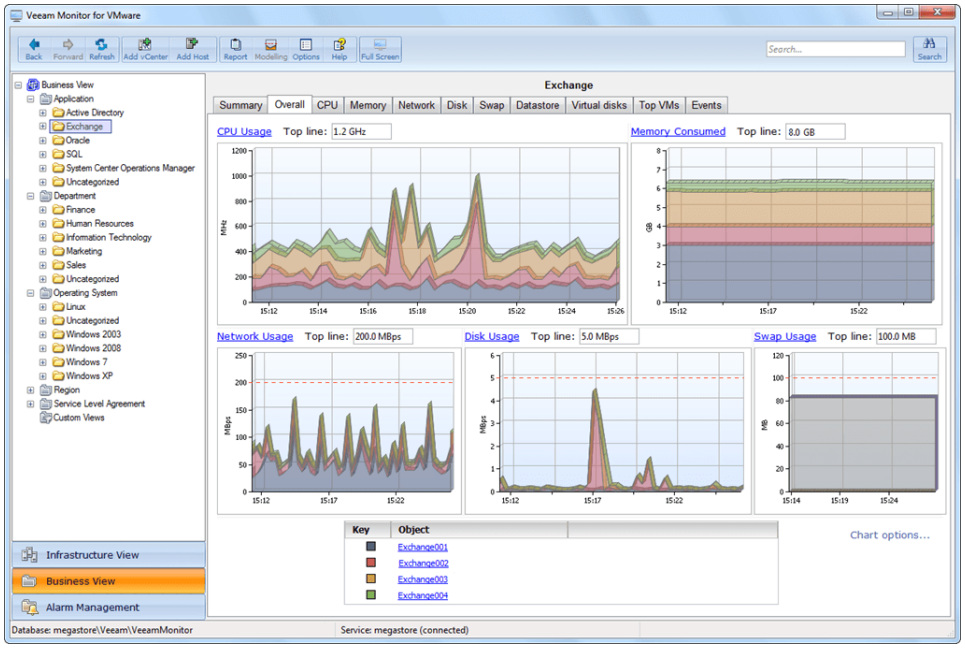


Obrázek 1.25: Kombinace několika technik v rámci vFogLight, včetně indikátorů, liniových a plošných grafů [34]

Jak můžeme z obrázku vidět, vFoglight pěkně kombinuje několik rozdílných vizualizačních technik, ať už jsou to červené/žluté/zelené indikátory celkového stavu systémů, nebo vlastní průběh zatížení v čase, který je zobrazován liniovým (CPU), případně složeným plošným grafem (Paměť). Nevýhodou je na první pohled poměrně malé využití prostoru pro jednotlivé grafy, je tedy potřebná interakce pro zobrazení detailu. Rovněž při větším množství VM je v menších grafech velmi zhoršená indikace hledaného signálu, respektive problému se systémem.

1.4.3 Veeam Monitor

Veeam Monitor od společnosti Veeam Software je poměrně robustní monitorovací nástroj, který například umožňuje využití lokální SQL Express databáze, popřípadě připojení k databázi vlastní. Zobrazuje jednotlivé grafy pro využívání nejdůležitějších zdrojů na serverech: procesor, paměť, disk a síť.

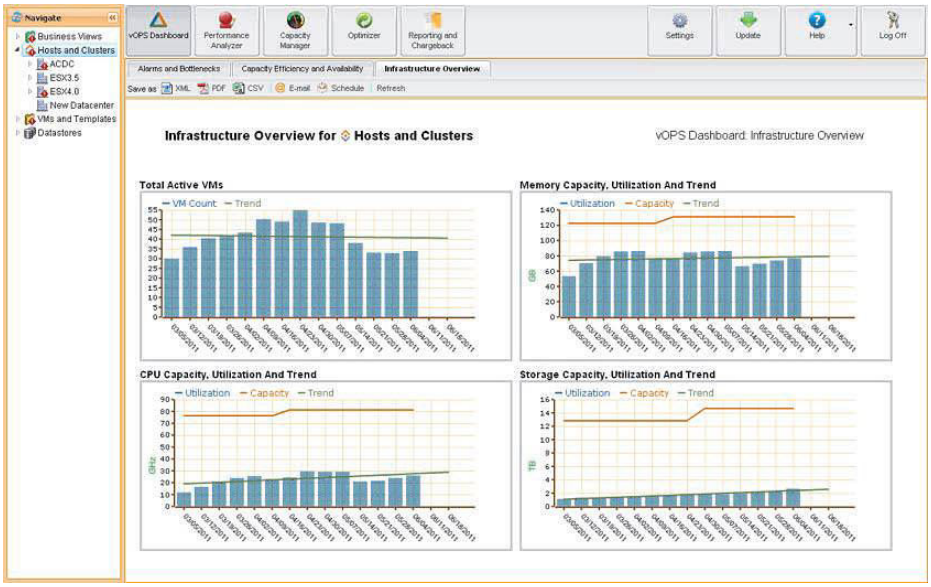


Obrázek 1.26: 2.5D Plošné grafy v rámci nástroje Veeam Monitor [37]

Z grafů opět poznáváme využití složeného plošného grafu pro všechny jednotlivé zdroje, v nichž jsou virtuální stroje rozlišeny barevným odstínem. Graf je rovněž vykreslován stylem 2.5D, což navozuje dojem určité plasticity. To však v některých případech může uživateli orientaci v grafu spíše zhoršit. Rovněž velké množství VM v takovém případě může v menších oknech zapříčinit horší čitelnost.

1.4.4 VKernel vOperations Suite

Společnost VKernel Corp. byla jednou z prvních společností s virtualizační kapacitou. Nástroj vOPS se používá jako virtuální zařízení, takže není vyžadována žádná licence OS nebo tradiční instalace aplikace/databáze. VKernel vOperations Suite obsahuje několik komponent, předně monitoring výkonu, správce kapacity, optimalizátor a reporting.

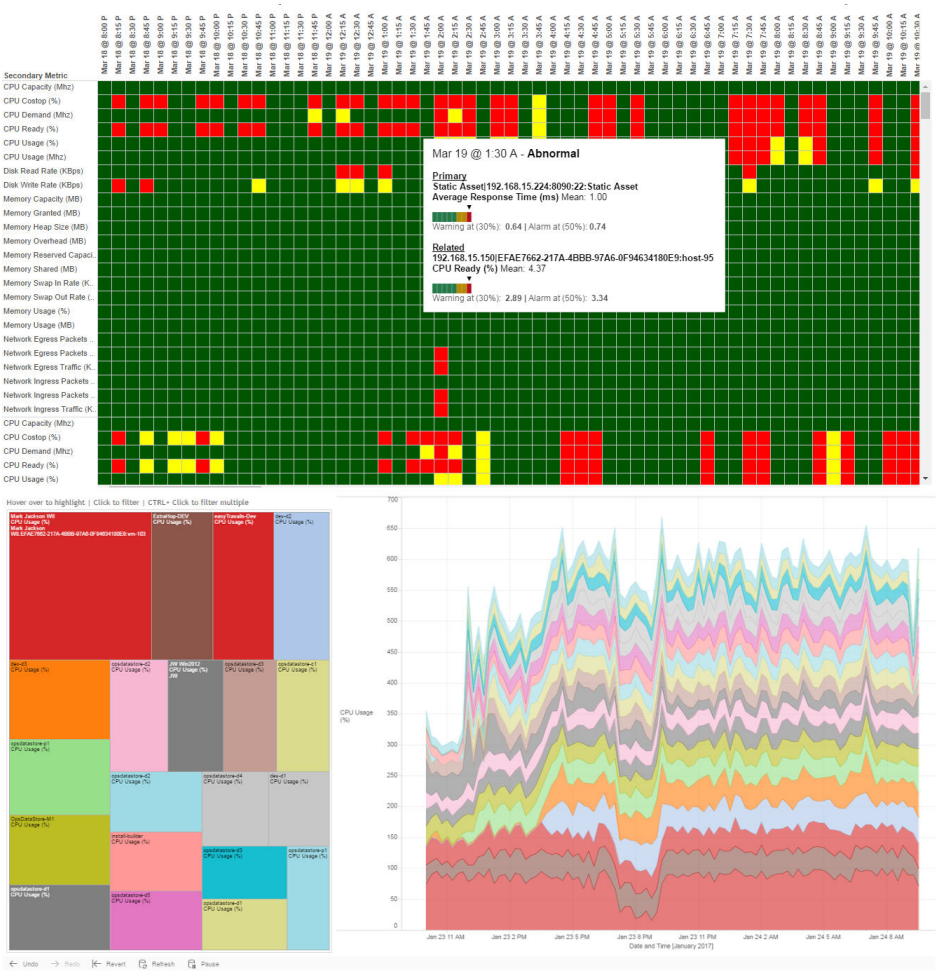


Obrázek 1.27: Kombinace histogramu a čárového grafu pro zobrazení tendencí v čase v rámci nástroje vOperations Suite [38]

V rámci vizualizace vOPS si můžeme povšimnout lehce změněného přístup oproti předchozím nástrojům; jakožto primární vizualizační technika pro zobrazování vytěžování je zde zvolena metoda sloupcového grafu, respektive histogram, a to v kombinaci se dvěma liniemi, z nichž jedna zobrazuje maximální kapacitu daného zdroje na serveru a druhá vývojový trend v čase. Tím se vyřeší nedostatek histogramu v rámci jeho optické diskretizace.

1.4.5 OpsDataStore

Nesmírně silným nástrojem je bezpochyby OpsDataStore. Ten jako takový neslouží prioritně jako monitorovací nástroj, jedna z jeho funkcí je však i možnost sběru dat a metrik z platforem jako je například VMware vSphere a následná možnost tato data vizualizovat. Umožňuje celou řadu interakcí uživatele s výběrem proměnných, časových intervalů a jednotlivých technik vizualizací, mezi kterými je krom liniových a plošných grafů využívána i heatmapa a treemapa, a to v souvislosti se zobrazováním určité metriky pro daný časový okamžik v případě heatmapy a rozložením vytíženosti serveru na jednotlivé VM v případě treemapy.

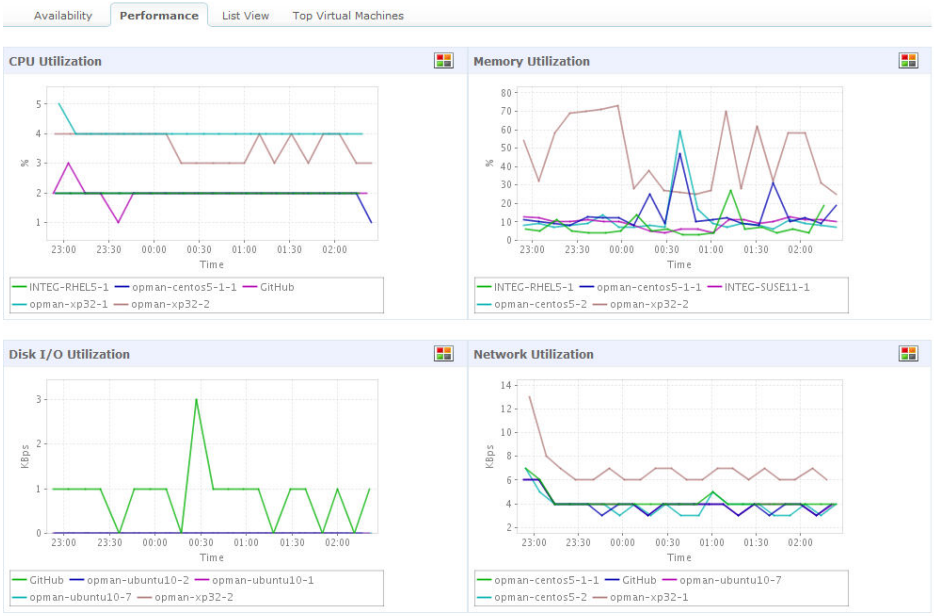


Obrázek 1.28: Heatmapa a kombinace treempay a složeného plošného grafu pro zobrazení procesorové vytíženosti jednotlivých VM daného serveru v rámci OpsDataStore[36]

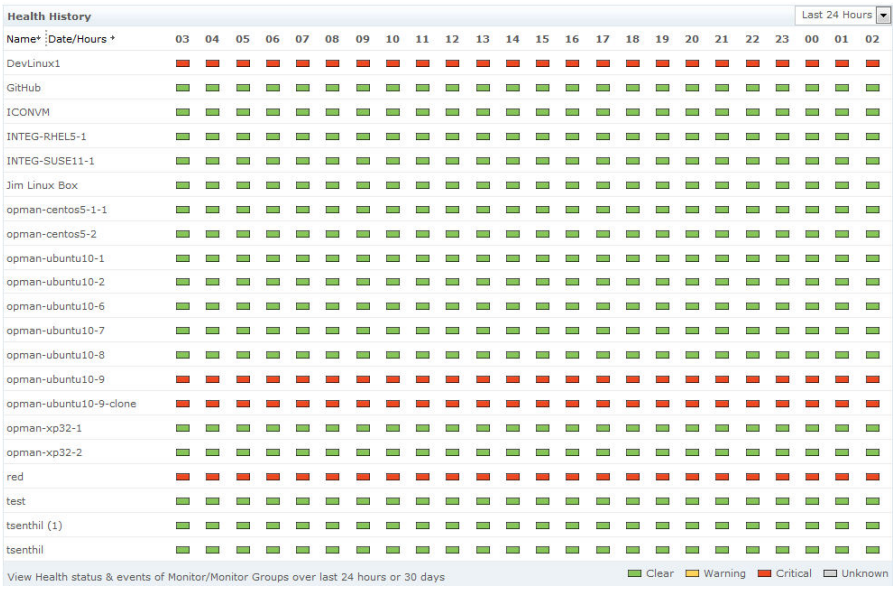
Z hlediska naší implementace nebude vyžadována takováto vysoká flexibilita (není tolik use casů) pro nejrůznější problematiku, respektive takto velké možností typů sledování. Přesto určitá kombinace jednotlivých technik pro konkrétní use case bude žádoucí a vyžadována. Rovněž si můžeme na obrázku výše všimnout problému rozeznatelnosti některých odstínů barev v případě plošného grafu.

1.4.6 ManageEngine

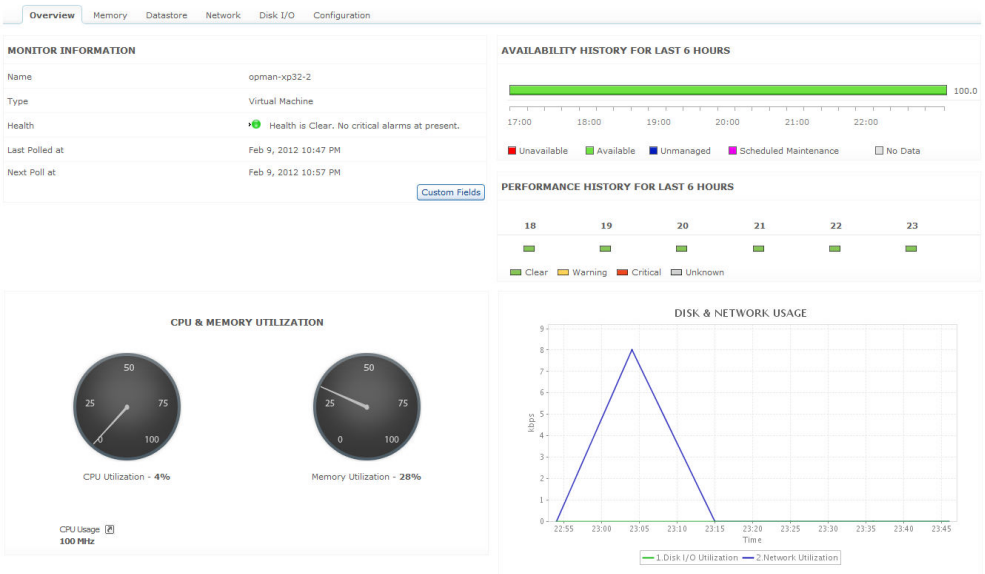
ManageEngine Applications Manager v rámci společnosti ManageEngine poskytuje komplexní výkonnostní metriky pro sledování VMware ESX/ESXi serverů a jejich VM. Správce aplikací se připojuje prostřednictví vlastního API a pomáhá monitorovat stav zdraví a výkonu hostitelských serverů a jejich odpovídajících virtuálních strojů.



Obrázek 1.29: Liniové grafy pro sledování trendů v čase v ManageEngine [39]



Obrázek 1.30: Heatmapa v rámci přehledu vytíženosti v ManageEngine [39]



Obrázek 1.31: Detailní pohled na konkrétní VM v ManageEngine [39]

V rámci vizualizace ManageEgine spoléhá zejména na liniové grafy společně s vizualizací detailů prostřednictvím tachometrů (pro zobrazení vytíženosti) a jednoduché heatmapy (pro vizualizaci vytíženosti VM v rámci určitého časového období).

1.4.7 Shrnutí existujících řešení

Mimo zmíněných nástrojů existuje pochopitelně nespočet dalších, které obdobně umožňují vizualizaci za účelem monitoringu jednotlivých částí serverového cloudu. V každém případě námi ukázané nástroje poměrně jasně ukazují trendy v oblasti vizualizačních technik pro zkoumané problémy. V prvé řadě to je tedy především složený plošný graf, který je občas doplněn či nahrazen liniovým grafem. Tento typ grafu umožňuje v rámci možností dobře zobrazovat změny v čase, trendy a rovněž kritické situace. Nevýhodou může být horší porovnávání několika kategorií (VM, serverů) mezi sebou v rámci několika časových okamžiků, a to právě v případě velkých výkyvů zatížení. Ty můžou být způsobeny například náhlým přidáním nového virtuální stroje/serveru, nebo rapidním nárůstem zatížení jednoho z nich. Za tímto účelem bude tedy vhodné pro zřetelnější zobrazení přesného poměru vytíženosti pro daný časový okamžik zobrazit jiný graf, respektive doplnit graf stávající o graf detailní, například složený sloupcový graf či kruhový diagram. Alternativou k plošným grafům mohou být grafy liniové, ty však s větším počtem sledovaných objektů výrazně rychleji ztrácí na čitelnosti.

Rovněž heatmapa je, zejména u robustnějších nástrojů, poměrně častou metodikou pro vizualizaci. Její využití je vhodné při dlouhodobém sledování jedné proměnné přes časová období v rámci několika objektů stejné kategorie (např. vytíženost všech serverů v rámci měsíce po přesně daných časových intervalech). Heatmapa rovněž umožňuje v našem případě poměrně rychle rozpoznat problém a to díky změně v intenzitě zvoleného odstínu (typicky bílá = nevyužito, sytě červená = vysoké vytížení, alternativně škála zelená-žlutá-červená).

V neposlední řadě mohou být uživatelsky přívětivé některé méně typické techniky, jakými jsou kupříkladu tachometry, které intuitivně mohou vyobrazovat vytížení jednotlivých serverů v daném časovém okamžiku.

Celkově je však i z jednotlivých existujících řešení evidentní, že bude s největší pravděpodobností potřeba kombinovat více vizualizačních technik, a to jak v závislosti na zvoleném cíli vizualizace, tak i v závislosti na hierarchickém vztahu mezi VM a servery, respektive neinterferujícími vztahy mezi skutečnými servery (zatíženost jednoho serveru by neměla ovlivňovat zatížení jiného).

1.5 Sběr a zpracování dat

Nyní se můžeme detailněji zaměřit na způsob získávání vizualizovaných dat, jejich uchovávání a na případné předzpracovávání před samotnou vizualizací. Valná většina této činnosti úzce souvisí s částí architektury výsledné aplikace, kterou můžeme obecně nazvat backendem, případně se samostatnou backendovou aplikací. Tato část obecně představuje hlavní výpočetní část nástroje, která se primárně stará právě o sběr a parsování dat (v našem pří-

padě procesorové a paměťové zatížení jednotlivých serverů a VM v různých časových okamžicích) a jejich další zpracování. Backend typicky pracuje odděleně od hlavní aplikace kvůli přílišné časové náročnosti jednotlivých operací s velkým množstvím dat. Cílem takovýchto operací poté bývá získávání relevantních informací, oddělení redundancí či irelevancí a případná agregace či jiná transformace, při které typicky dochází k výraznému zmenšení objemu zpracovávaných dat. Hlavní aplikaci, jež se v našem případě bude primárně soustředit na vizualizaci, jsou tedy předávaná data už celkově předzpracována a často značně redukovaná, což zajistí její plynulejší funkčnost.

Základní požadovanou funkcionalitu backendové části můžeme rozdělit do následujících bodů:

- Získání a přenos dat
  Backendová část aplikace typicky různými prostředky získává data od zdroje, kterým je v našem případě cloud, respektive server. Backend tedy v určitých intervalech vysílá požadavky na odeslání aktuálních dat, kterou jsou mu poté v případě úspěšné autorizace odeslány. Získávání dat může být rovněž spojeno s dedikovaným sběračem dat, jež je reprezentován dotazovacím deamonem, jež neustále běží na zdrojovém serveru, získává příslušná data a celkově pak pracuje nezávisle na zbytku aplikace. Backendová část se v takovém případě dotazuje na tohoto deamona a přenáší získaná data k sobě pro další zpracování.

- Zpracování dat
  Veškerá získaná data je obvykle nutné před uložením zpracovat. Způsob zpracování je definován nejčastěji typem dat, požadavkem na jejich následující využití a omezením velikosti databáze. Typicky se tedy jedná o určitou formu agregace dat, úpravu formátu, či odstranění zbytečných záznamů. V rámci zpracování může dojít i k přidání dodatečných informací, je-li to nezbytné pro další chod aplikace. Některé typické operace s daty lze poté rovněž řešit za pomocí existujících nástrojů.

- Uložení dat do databáze
  Backendová část aplikace úzce spolupracuje s databází a je tedy logicky nezbytné, aby tato uměla s příslušným typem databáze a jejími technologiemi pracovat. Obvykle se vyžaduje schopnost ukládání dat do databáze a v případě nutnosti pak i schopnost z databáze data číst (ověření platnosti dat, detekce chyb).

- Načtení dat do hlavní aplikace
  V závislosti na zvolené architektuře aplikace je možné, že backendová část bude muset přímo zasílat data do hlavní aplikace pro vizualizaci (typická varianta). V takovém případě je nutné data z databáze přečíst

a přeposlat k vizualizaci, případně provést ještě nějakou dodatečnou transformaci či úpravu před finální vizualizací.

1.5.1 Rozlišení dat

Pod pojmem rozlišení dat se chápe především množství získaných dat, přičemž vyšší rozlišení obecně představuje detailnější a přesnější data, která jsou však proporcionálně náročnější na zpracování a mohou tak významně zpomalit/-zrychlit používání aplikace, respektive přístup k datům. Nalezení správného kompromisu při volbě vhodného rozlišení pro aplikaci je rozhodující pro správnou funkčnost výsledné aplikace.
Mezi základní přístupy při volbě rozlišení dat řadíme následující:

- Maximální rozlišení dat
  Naivní přístup, kdy veškerá data uchováváme tak, v jakém stavu je získáme. Nedochází tedy k žádné úpravě ani redukci objemu, což zapříčiňuje nejvyšší míru svobody co se manipulace s daty týče, kdy uživatel bude moci detailně sledovat přesná zatížení jednotlivých částí cloudu tak, jak byla pro dané časové okamžiky naměřena. Tato výhoda je však podmíněna obrovským množstvím dat, jež je třeba uchovávat a s kterými je třeba při běžném používání aplikace manipulovat. V závislosti na množství VM a serverů v cloudu tak můžeme, společně se zvolenou frekvencí dotazování se na zatížení, velmi rychle dojít i k několika miliardám datových bodů, což značně zpomaluje práci s výsledným nástrojem. V případě dlouhodobějšího skladování dat pak můžeme velmi rychle dosáhnout limitů databáze.

- Agregace dat
  Nejtypičtější alternativou k naivnímu uchovávání všech dat je určitá forma agregace. Ta sice částečně zneplatní získaná data, neboť je aplikováním specifické metody zkreslí, avšak značně redukuje objem dat, až na námi zvolenou velikost v závislosti na velikosti agregace. V našem případě se agregace bude moci provádět pouze nad časem, tedy pro určitý časový interval se provede sjednocení všech zátěžových dat u časových bodů v tomto intervalu do jednoho či několika málo údajů. Agregace přes serverové zatížení či zatížení jednotlivých VM do značné míry postrádá smysl.
  Mezi základní metody agregace patří pak následující:

  – Průměr

  – Součet

  - Medián
  - Maximum
  - Minimum
  - Rozptyl

  V rámci našeho řešení budeme při použití agregace primárně využívat průměru, s možností zobrazení maxima a minima v případě sledování dlouhodobějších trendů.

- Vzorkování dat
  Oproti agregaci, která zpracovává všechna data, nad kterými poté provádí specifickou agregační operaci, vzorkování (angl. sampling) vybírá vzorky z celkového množství dat na základě specifických požadavků a následné agregační operace provádí striktně s těmito vybranými částmi dat. Tím dochází k ještě radikálnější redukci objemu, přičemž získaná informace může ztratit na vypovídající hodnotě jen velmi málo. Kvalita samplingu poté závisí striktně na vybrané metodě při získávání vzorků, kdy se obvykle používají pravděpodobností přístupy za použití náhodných čísel, které korespondují s vybranými datovými body z celkové množiny, což zajistí eliminaci případné korelace mezi vybranými vzorky. Všeobecně však platí, že vzorkování není vhodný postup v případě monitoringu dlouhotrvajících procesů (jeho síla je spíše v objevení vzorů a prediktivním modelování), což jeho použití v rámci našeho řešení značně znevýhodňuje.

1.5.2 Databáze

Jednou z důležitých činností v rámci manipulací s daty je pochopitelně i jejich dlouhodobější uchovávání. Za tímto cílem je vhodné použití databáze, která obecně představuje skladiště dat, kde jsou uložena připravená data. Díky velkému objemu našich dat a možnosti měnit časové rozsahy i směrem do minulosti je ukládání dat do databáze nezbytné pro plynulý běh aplikace.

Existuje velké množství databázových modelů, které dále specifikují způsob ukládání dat a jejich logickou strukturu a organizaci v rámci databáze, popřípadě jakým způsobem můžeme s daty v databázi manipulovat. Nejčastěji používaným modelem je relační model a k němu přidružené relační databáze (někdy rovněž označovány jako SQL databáze), které používají tabulkový formát ukládání dat. Mezi další používané databázové modely (často ne zcela přesně označovány za NoSQL databáze) patří grafové modely a přidružené

grafové databáze, MultiValue modely či objektově-orientované modely a databáze, které se inspirují v objektově-orientovaných programovacích jazycích.

Za účelem vhodné volby pro databázové řešení je třeba si uvědomit, jaký charakter naše data budou primárně mít. Kromě jednotlivých zatížení, které lze částečně hierarchicky uspořádat (zatížení serveru je cca rovno součtu zatížení vlastních VM, které běží na daném serveru), lze předpokládat shlukování dat do specifických skupin, nejčastěji v závislosti na zvoleném časovém intervalu, popřípadě jiné sdílené charakteristice. Tedy je pravděpodobné, že jednotlivé dotazy na databázi budou ve většině případu vracet odpovědi typicky spíše většího objemu.

Nyní ve zkratce zmíníme několik základních možných variant pro volbu databáze:

- Relační databáze
  Tradiční typ databáze, jež je v současné době stále nejrozšířenější používanou databází. Díky své robustnosti (striktně dodržuje ACID), jednoduchosti a pochopitelnosti je lze použít ve vztahu k téměř jakémukoliv problému. Základem relační databáze jsou databázové tabulky, které lze chápat jako entity mapující prvky skutečného světa. Tabulky se skládají ze sloupců a řádků, přičemž sloupce reprezentují atributy, řádky pak jednotlivé záznamy. Mezi jednotlivými entitami se pak vyskytují vazby s různou kardinalitou a stupněm.

  Relační databáze můžeme považovat za jakýsi zdravý standard, který však v současné době v určitých aspektech zaostává. Kupříkladu roztříštěnost dat do jednotlivých tabulek zapříčiňuje pomalejší dotazování v rámci větších objemů dat, kdy se tabulky musí pro každý individuální dotaz opětovně sestavovat. Je tedy otázkou, do jaké míry funkcionalita klasické relační databáze bude stačit v případě našich dat.

- Wide column databáze
  Tento typ databáze se v lecčem podobá klasické relační databázi, avšak jednotlivé řádky v rámci jedné tabulky mohou mít jinou množinu atributů, tedy sloupců. Jednotlivé záznamy pak v rámci jedné entity mají často zcela odlišné vlastnosti a hodnoty, což značně zmenšuje nutnou velikost výsledné databáze. Navzdory tomu je vyhledávání v databázi díky nedodržování zásad konzistence omezené, na druhou stranu však možnost vysoké horizontální škálovatelnosti takovéto databáze ji poměrně zvýhodňuje, zejména při použití v rámci webových technologií.

- Dokumentová databáze
  Jak již název napovídá, dokumentová databáze ukládá datové údaje do dokumentů, respektive do kolekcí souborů, které obsahují vazby typu klíč-hodnota, přičemž klíč může být explicitně definován, nebo se jedná o metadata získaná z konkrétních dat. Formát uspořádání klíč-hodnota vazeb pak specifikuje konkrétní typy dokumentových databází, které mj.

zahrnují XML databáze, grafové databáze, případně jiné. Obecně dokumentové databáze nejčastěji využívají formáty XML, YAML, JSON a BSON. Velkou výhodou jsou pak poměrně uvolněnější pravidla v rámci používání databáze a ukládání dat do ní, což umožňuje vysokou flexibilitu takovéhoto systému.

- Distribuovaná databáze
  Podstata distribuovaných databází spočívá v uložení velkého množství dat mezi větší počet samostatných počítačů, jež nemusí být nutně nijak specializované. Zpracování takto uložených dat pak probíhá paralelně, kdy je zadaná úloha (dotaz), nejprve rozdělena na pod-úlohy, ty jsou poté vykonávány paralelně na více uzlech (počítačích) naráz, následně jsou vyprodukovány dílčí výsledky (odpovědi na dotaz), které poté spojíme do celkového výsledku. Tento princip se nazývá MapReduce, kde nejprve mapujeme úlohu na jednotlivé uzly (rozdělení a distribuce) a následně redukujeme dílčí odpovědi do jedné finální.

    Velkou výhodou distribuované databáze je schopnost zpracovávání enormního množství dat, naproti tomu zásadní nevýhodou je vysoká komplexnost a celková velikost takovéhoto řešení, která by v rámci naší problematiky pravděpodobně přesáhla únosnou mez.

Důležitým faktem při zpracovávání velkého množství dat je i trvanlivost těchto dat v rámci databáze, jinými slovy jak dlouho je budeme uchovávat. Pokud budeme chtít monitorovat trendy v rámci větších časových období (let), bude potřeba mít tato data přirozeně uložená pro celé takovéto období, což může vést k problémům s limity databáze. V opačném případě stačí v pravidelných intervalech databázi promazávat, respektive data s určitým stářím (od určitého timestampu dál) z databáze odstranit, čímž se celkově sníží množství dat, které se bude v daném okamžiku v databázi nacházet.

1.6 Výsledky analýzy

Nyní na závěr kapitoly shrneme zásadní poznatky, které analýza přinesla, a na jejichž základě budeme dále postupovat při vytváření návrhu a následné implementace aplikace.

Jak analýza vizualizačních technik, tak analýza existujících řešení poměrně jasně určila, kterým směrem se v rámci volby vizualizačních technik vydat. V současné chvíli se jako nejlepší postup jeví tedy kombinace heatmapy, která bude použita striktně pro vizualizaci zatížení jednotlivých serverů v rámci overview. Dále po zvolení časového intervalu a konkrétního supervizora dojde k zobrazení preview, kde se bude jednat již o složený plošný graf v kombinaci s jednoduchým plošným grafem pro vyobrazení okolního kontextu daného supervizora. Pro zobrazení detailu je poté vybrán rovněž složený plošný graf, kde jednotlivé plochy odpovídají konkrétním VM. V rámci detailu poté bude

uživateli umožněna interakce při volbě prioritního VM, jež se zarovná vůči ose X. Dále bude uživatel mít možnost v rámci detailního grafu zoomovat a rovněž vybírat konkrétní časové okamžiky pro "detail detailu". Tyto konkrétní okamžiky budou vizualizovány v dalším grafu jakožto složené sloupcové grafy, které mají v rámci diskrétních hodnot lepší čitelnost než spojitý plošný graf. Rovněž zde bude mít uživatel možnost zoomu, což umožní komplexní navigaci ve vizualizaci a celkově umožní možnosti podrobnější orientace v rámci naší problematiky.

Co se týká sběru a zpracování dat, v současné chvíli se jeví jako nejpravděpodobnější použití deamona, který se bude v rámci cloudu v pravidelných časových intervalech dotazovat prostřednictvím REST API na jednotlivá zatížení všech VM a jednotlivých serverů. Tato data budou poté uložena do databáze. Při vybrání konkrétního časového intervalu pro monitoring budou poté příslušná data z databáze přečtena, dále prostřednictvím agregace dojde k jejich redukci a takto redukovaná data budou poslána do hlavní aplikace k vizualizaci. V případě změny rozsahu či větším detailu může dojít k opětovné agregaci nad menším intervalem, což bude pravděpodobně vyžadovat opětovné přečtení dat z databáze. Alternativou tedy může být uchovávání všech dat v rámci intervalu přímo v aplikaci a agregaci provádět v závislosti na zvoleném detailnějším pohledu. Toto řešení však pravděpodobně může mít neblahý vliv na rychlost aplikace, neboť tato činnost může být poměrně časově náročná. Řešením do budoucna tedy může být nasazení speciálního aplikačního serveru, kde se tyto náročnější operace s daty budou odehrávat.

Kapitola 2

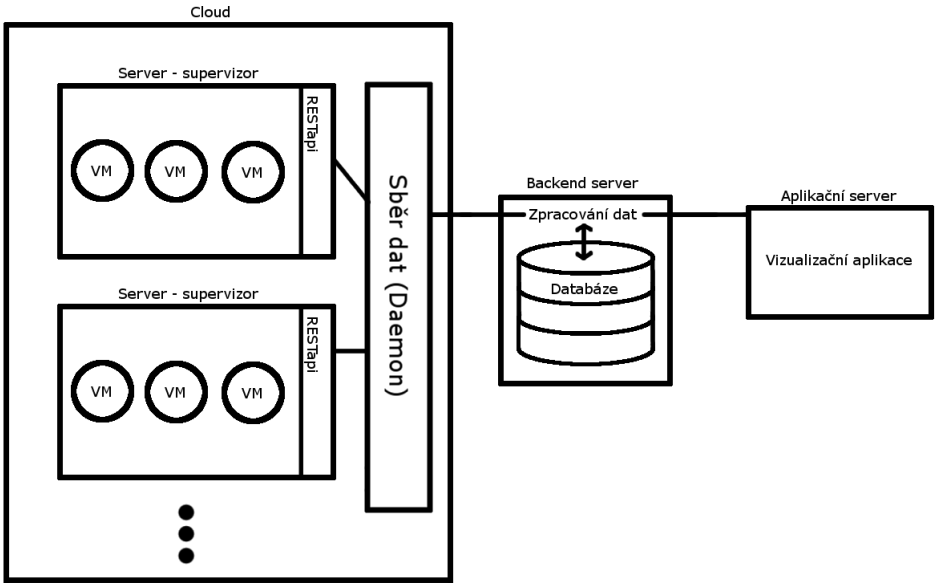
Návrh

Dosavadní analytická část této práce se věnovala průzkumu velkého množství přístupů a možností stran jednotlivých vizualizačních technik a vlastních požadavků na vizualizaci a aplikaci jako takovou, včetně zvažování zásadních přínosů a nevýhod potenciálních řešení. Kapitola návrhu se plně věnuje konkrétním návrhům jednotlivých částí aplikace včetně uživatelského rozhraní a několika zásadním algoritmům, které bude vhodné pro dílčí funkcionality aplikace implementovat. Z hlediska návrhu uživatelského rozhraní jsou poté nejdůležitější wireframy, které vyobrazují jednotlivé částí grafického rozhraní celkové aplikace. Součástí práce na návrhu aplikace bude i implementace jednoduchého funkčního iniciálního prototypu, jehož hlavním úkolem je otestovat kvalitu vybraných vizualizačních technik a některých zjištěných poznatků z předchozí kapitoly, respektive odhalit potenciální nedostatky jednotlivých částí. Výsledný prototyp bude poté dále použit jako výchozí pozice pro další vývoj aplikace jako takové.

2.1 Návrh architektury aplikace

Kapitola návrhu celkové architektury výsledné aplikace částečně navazuje na závěrečnou kapitolu analýzy zabývající se sběrem dat1.5 . V této části se tedy podíváme na potenciální strukturu výsledné aplikace, na jednotlivé základní části a moduly, které výsledná aplikace bude obsahovat a na jejich jednotlivá specifika.

Základní návrh struktury aplikace se dá rozdělit do tří relativně autonomních částí, které však spolu neustále komunikují a jež můžeme vidět znázorněny na diagramu níže.



Obrázek 2.1: Diagram návrhu architektury aplikace

Předně se tedy jedná o vizualizační aplikaci, jejíž úkolem je získaná data prostřednictvím vybraných postupů a technik vizualizovat a která je rovněž spojena s konkrétním uživatelským rozhraním, jež by mělo být opticky přívětivé a jednoduché na ovládání. Tento modul představuje hlavní komunikační kanál s uživatelem, kde uživatel volí rozsah vizualizovaných dat a celkově interaguje s vytvořenou vizualizací a snaží se jejím prostřednictvím dosáhnout specifických cílů (monitorování, sledování trendů apod.). Mimo jiné by tato část aplikace rovněž měla obsahovat veškeré nástroje pro interakci, tedy filtraci, potažmo specifickou formu transformace dat a animace jednotlivých vizuálních změn. Tuto část lze tedy považovat za hlavní a z tohoto důvodu je jí rovněž v rámci této diplomové práce věnována drtivá většina pozornosti, jak ze strany analýzy (vizte kapitolu 1), tak implementace.

Druhou částí aplikace je backendová část, tedy část, jež se primárně zabývá zpracováním získaných dat, ukládáním do přidružené databáze a následné zasílání specifické části dat (na základě vybraného časového intervalu a po provedení agregace) do hlavní části aplikace pro vizualizaci těchto dat. Oddělení této části je obecně uznávanou praxí zejména z důvodu větší náročnosti jednotlivých specifických operací nad velkým množstvím získaných a ukládaných dat. V případě, že by tyto procesy datové manipulace byly přímou součástí hlavní vizualizační částí aplikace, mohlo by dojít k významnému zpomalení celkového systému a do značné míry tedy znemožnění jeho plynulého používání. Z tohoto důvodu je také často backendová část aplikace dedikována

na vlastní server, právě za účelem odstínění vzniklého zatížení mimo hlavní část aplikace. Přidružená databáze, jež je s touto částí úzce spojená, pak klade důraz na schopnosti backendu stran komunikace s daným typem použité databáze, ať už se jedná o ukládání, tedy zápis, či o čtení uložených dat a průběžné mazání nevalidních či zastaralých záznamů.

Poslední, neméně důležitou částí aplikace, je sběrač dat, který se nachází přímo na cloudovém systému. Tento sběrač je obvykle reprezentován daemonem, tedy konkrétním programem, jež v rámci cloudu nepřetržitě běží a ve zvolených pravidelných intervalech se dotazuje prostřednictvím dedikovaného REST API (viz. níže) na zatížení (paměťové a procesorové) jednotlivých hostujících serverů (supervizorů), respektive na zatížení všech konkrétních VM, které na daném serveru v daném okamžik existují. Takto získaná data pak v konkrétním formátu zasílají na backendovou část, kde dojde k jejich zpracování a uložení do databáze.

2.1.1 Platforma

Velmi důležitou volbou z hlediska návrhu budoucí aplikace a dílčích částí její architektury je nepochybně volba platformy, na které se bude výsledný systém nacházet, respektive základní forma výsledného programu. Předně je vhodné se na začátek omezit čistě na počítačové prostředí, tedy vynechávat mobilní platformu jako takovou, přestože tato v současné době představuje velké možnosti a v zásadě by umožňovala vysokou přístupnost a možnosti využívání systému (možnost sledování zatížení prakticky odkudkoliv, výrazné zvýšení použitelnosti). Nasazení aplikace na mobilní platformy je tedy v zásadě žádoucí a může být předmětem postupného rozšiřování systému do budoucna.

Při omezení se na počítačovou platformu máme v zásadě dvě základní možnosti, jakým způsobem vytvářet výslednou aplikaci:

- **Desktopová aplikace**
  Desktopová (nativní) aplikace představuje řešení formou programu, jež prostřednictvím vybraného operačního systému běží přímo na hostitelském počítači, jehož paměť a procesor je za tímto účelem využíván. Za desktopové aplikace se často považují v současné době i ty, jež do značné míry používají webové technologie a internet při komunikaci jednotlivých částí systému, ale jejichž hlavní část je stále na bázi klasické desktopové aplikace (kupříkladu klasické počítačové hry považujeme stále za desktopové aplikace, nehledě na to, že často již dnes přímo vyžadují připojení k internetu).

  Základní výhodou tohoto přístupu je předně možnost využití výchozích vlastností konkrétních operačních či strojových systémů, což zapříčiňuje jednodušší práci s výslednou aplikací a obecně rychlejší běh programu. Jejich vývoj je pak obecně spojen s větším množstvím vývojářských nástrojů a SDK, což značně urychluje jejich vytváření a čás-

tečně zajišťuje robustnost a bezpečnost výsledku. Naproti tomu hlavní nevýhody představují často větší cena ze strany údržby výsledného produktu, větší možnosti zásahu do programového chodu aplikace a potenciálně tak zjištění nežádoucích informací (průmyslová špionáž atd.). Rovněž ukládaná data nejsou často žádným způsobem zálohovaná (pokud se nepracuje s větším objemem dat je řešení prostřednictvím databázových systémů často opomíjeno) a při poškození OS nebo části hardwaru může dojít k nenávratné ztrátě. Tento nedostatek se pak často řeší právě prostřednictvím využití určité části webových systémů (např. cloudu) i v rámci tohoto typu aplikací.

Přístup k implementaci a výsledná použitelnost desktopové aplikace pak rovněž nepřímo souvisí s vybraným operačním systémem. OS Linux typicky usnadňuje zkušenému vývojáři proces vývoje prostřednictvím své otevřenosti a přívětivosti, kde jednotlivé nástroje často bývají pod otevřenou licencí a jsou navzájem vysoce kompatibilní. Nevýhodou je pak menší rozšířenost těchto OS a často kritizovaná vyšší složitost běžného používání. Microsoft Windows je oproti tomu v současné době nejrozšířenější operační systém na světě a aplikace vyvíjené na této platformě mívají k dispozici velké množství knihoven a kvalitních vývojářských nástrojů. Tyto jsou jsou však často placené. Další nevýhodou pak může být i určitá uzavřenost systému a nekompatibilita s některými vybranými prostředky.

- **Webová aplikace**
  Webová aplikace je poskytována uživatelům z konkrétního webového serveru skrze internetovou síť, potažmo webového prohlížeče jakožto klienta. Webové aplikace se vyznačují především vysokou mírou rozšiřitelnost a nezávislosti na operačním systému či stroji, na němž bude aplikace spouštěna. Webová aplikace pak v současné době částečně zaručuje možnost použití i na mobilních platformách. Výhodou je i vysoká flexibilita vývoje, kdy jednotlivé programovací jazyky často disponují velkou řadou knihoven, jež jsou zejména v souvislosti s vizualizačními procesy poměrně robustní a poskytují velkou řadu knihovních volání, jež umožňují značně zjednodušit implementaci složitých vizualizačních přístupů (animace, interakce atd.). Drtivá většina dat je pak rovněž implicitně uložena na vzdáleném serveru, což napomáhá při potenciální ztrátě.

  Hlavní nevýhodou je pochopitelně nutnost internetového připojení, bez něhož nelze webovou aplikaci spustit. Na druhou stranu, internetové připojení je dnes hojně dostupné a i velká část desktopových aplikací toto přípojení v současné době přímo či nepřímo vyžaduje. Webové aplikace jsou taktéž zranitelnější ze strany potenciálního útoku třetích stran a následnému úniku důvěrných dat. V neposlední řadě webové aplikace stále trpí na všeobecně horší kvalitu výsledného produktu, což je způsobeno omezeným přístupem ke zdrojům, přestože se tento problém s

rozvojem webových technologií daří postupně eliminovat.

- **Hybridní aplikace**
  Jak již název název napovídá, hybridní aplikace jsou ty, jež v zásadě ve vyrovnaném poměru využívají jak elementy nativních desktopových aplikací, tak aplikací webových. Díky tomuto přístupu je možné spojit jednotlivé výhody obou přístupů a při chytrém využití i navzájem eliminovat dílčí nedostatky. Hybridní aplikace v současné době zažívají postupný vzestup a lze tedy do budoucna předpokládat daleko častější využití hybridních strategií při obecném návrhu aplikace. V zásadě jedinou, o to však větší nevýhodou je stále ještě velká náročnost při implementaci takovýchto systémů, kdy je třeba kombinovat technologie webové i desktopové aplikace a správné řešení tedy vyžaduje znalosti stran návrhu a implementace obou těchto systémů na poměrně detailní úrovni.

Všechny tyto varianty jsou v rámci naší problematiky v zásadě platnou možností. Hlavním aspektem při rozhodování o volbě zvolené platformy pak v našem případě byla především menší zkušenost s implementačním procesem webové aplikace a s ní spojenými nástroji a jazyky, tedy hlavní vizualizační část bude implementována formou desktopové aplikace. Je však třeba uznat, že webová (potažmo hybridní) aplikace obecně významně rozšiřuje použitelnost, respektive přístupnost výsledného vizualizačního systému. Tedy obdobně jako v případě rozšíření systému na mobilní platformy, vytvoření alternativy formou webové aplikace, respektive aplikace hybridní, jež bude rovněž přímo propojena s paralelně vznikajícím systémem Pavla Podaného zabývajícího se vizualizací toků dat, bude s největší pravděpodobností předmětem navazujícího procesu rozšiřování naší aplikace.

Na závěr je rovněž nezbytné zmínit, že jednotlivé části architektury obecně mohou pracovat v rámci jistých omezení s rozdílnými platformami, byť tato kombinace může přinášet i dílčí problémy. V našem případě například z návrhu architektury vidíme, že díky podstatě webového cloudu není logicky možné webovou komunikaci zcela eliminovat a je tedy třeba jí v určitých částí systému použít (viz. např. zmiňované REST API), např. mezi backendem a sběračem dat, případně mezi hlavní vizualizační aplikací a backendem. Obecně však stále můžeme zvolené řešení považovat za aplikaci desktopovou, neboť většina implementace je právě soustředěna do zmíněné vizualizační části aplikace.

2.1.2 Návrh REST API

Technologie REST (Representational state transfer) je specifická architektura rozhraní, která je navržena pro distribuované prostředí a umožňuje prostřednictvím specifických HTTP požadavků snadný a jednotný přístup k vybraným zdrojům (konkrétní data, stavy aplikace, atd.), jež jsou identifikovány prostřednictví URI. Jelikož v současné chvíli systém BigCloud hojně REST API

využívá při komunikaci jednotlivých částí vlastního distribuovaného sytému, je využití této architektury pro sběr námi požadovaných dat k vizualizaci nejlepší možnou variantou.

Jelikož v našem případě budeme z cloudu, respektive z konkrétních server, data pouze číst a nikoliv modifikovat, budeme při návrhu REST API potřebovat pouze základní metodu GET, a to právě k získání konkrétního zdroje (dat). Zbylé metody POST, DELETE a PUT tedy využívat nebudeme. Formát odpovědi poté bývá typicky JSON (JavaScript Object Notation), který je v této souvislosti považován za standard a je lehce zpracovatelný a čitelný.

Základní návrh jednotlivých dotazů a odpovědí REST API je tedy následující:

1. `GET /api/info`

   Na základě tohoto dotazu budou vráceny pro nás relevantní informace o cloudu jako takovém, tedy počet serverů (supervizorů) a jejich konkrétní ID, pomocí kterých se bude potom moci dále dotazovat na konkrétní servery zvlášť. Pokud tyto informace budeme znát předem, případně je budeme moci získat z jiného, již existujícího zdroje, je tento dotaz redundantní a nebude tedy použit. Rovněž, pokud nepředpokládáme, že se v průběhu času bude počet serverů měnit, respektive že se tyto budou měnit pouze ve výjimečných případech, není tento dotaz potřebný, neboť nám stačí si zapamatovat ID jednotlivých serverů pouze jednou a dále pak s těmito pracovat (je možné je mít uložené v rámci sběrače/daemona). V opačném případě bude mít odpověď na tento dotaz následující formát:

```
{
"serv_number" : number of active servers
"servers" : [ server1-id,server2-id, ... ,last-server-id ]
}
```

2. `GET /api/servers/server-id`

   Tento dotaz směřuje již na konkrétní server, jež je jednoznačně určen prostřednictvím svého ID. Na základě tohoto dotazu získáme informace o velikosti pamětí a výkonu procesoru daného serveru, které budeme poté používat při vizualizaci jako horní hranici osy zatížení v jednotlivých grafech. Opět zde platí premisa, že pokud se tyto informace nebudou často měnit, je tato část dotazu v zásadě redundantní. Stačí pak abychom si tyto dílčí údaje o serverech někam explicitně uložili a poté je využívali. Důležitější je druhá část dotazu, která nám vrátí přesný počet VM, které v daném okamžiku na serveru jsou a následně i jejich

unikátní identifikátory. Prostřednictvím těchto se následně budeme, obdobně jako v případě serverů, dotazovat přímo na konkrétní VM a jeho atributy. Odpověď má tedy následující formát:

```
{
"S_memory" : server-memory
"S_processor" : server-processor
"virt_number" : number of active virtuals on this server
"virtuals" : [ virtual1-id,virtual2-id, ..., last-virtual-id ]
}
```

3. `GET /api/servers/server-id/virtuals/virtual-id`

   Tento dotaz vrátí jako odpověď analogicky informace stran konkrétního VM na daném serveru. Předně je to tedy velikost virtuální paměti a výkon virtuálního procesoru, jež jsou explicitně přiřazeny danému VM. Zákonitě pak tedy musí tyto hodnoty být menší než skutečná paměť, respektive procesor, jež jsou vztaženy k příslušnému supervizoru. Pro možnost porovnání je poté vhodné, aby tyto hodnoty byly společně s hodnotami serveru ve stejných jednotkách (GB, GHz, případně jádra aj.), alternativně aby jednotky byly rovněž součástí daného formátu. Druhou částí odpovědi jsou poté už skutečné hodnoty aktuálního zatížení, které mohou být buďto absolutní (konkrétní hodnota), či relativní (procentuální zatížení vzhledem k maximu). Při vizualizaci však obecně budeme spíše pracovat s relativními hodnotami vztaženými k danému serveru, tedy jiné vyjádření budeme přepočítávat. Formát výsledné odpovědi je pak tedy následující:

```
{
"V_memory" : virtual-memory
"V_processor" : virtual-processor
"mem_usage" : current virtual memory usage
"proc_usage" : current virtual processor usage
}
```

Vzhledem k tomu, že pravděpodobně nebude potřeba ptát se v daném okamžiku pouze na konkrétní server či VM, ale spíše se budeme naráz dotazovat na zatížení každé dílčí části cloudu, je otázkou, zdali v rámci praktičnosti není lepším řešením vytvořit pouze jeden jediný dotaz prostřednictvím metody GET, který by postupně vrátil odpovědi obsahující zatížení všech těchto jednotlivých částí (tedy všech serverů a všech jejich VM). Při zasílání získaných dat ze sběrače na backend je rovněž potřeba tato označit časovým razítkem (timestamp), které bude reprezentovat okamžik, pro který jsou získaná data platná. Tuto činnost pak rovněž bude s největší pravděpodobností vykonávat deamon, a to na základě systémového času cloudu.

2.2 Návrh uživatelského rozhraní

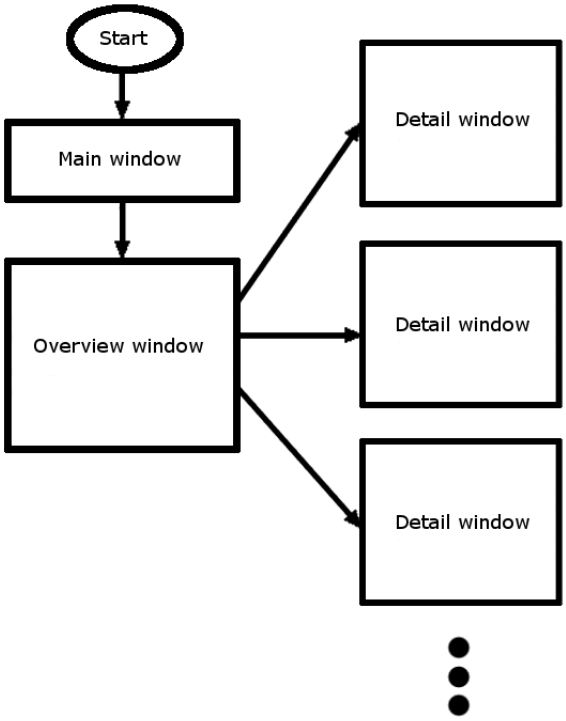
Navzdory tomu, že se zprvu může zdát, že kvalita uživatelského rozhraní není v rámci řešení naší problematiky významně určujícím aspektem, opak je pravdou. Nekvalitní aplikační rozhraní totiž může do značné míry sabotovat přínosy zvolených vizualizačních technik, a to zejména tím, že uživateli nebude jasné, jakým způsobem daná technika funguje a co je jejím účelem, respektive jaké možnosti mu dává či jakým způsobem se aplikace ovládá. V tomto směru je vhodné dbát na dodržování konvencí a základních zásad při vytváření obecného uživatelského rozhraní. Mezi tyto základní principy návrhu UI patří pak následující:

- Efektivnost
  Efektivní a jednoduché řešení sledovaných problematik
- Flexibilita
  Možnost přizpůsobení řešení konkrétnímu uživateli a dané úloze
- Konzistence
  Možnost opakovaného použití získané znalosti z používání systému, snaha obdobné problémy řešit unifikovaně
- Vizuální hierarchie
  Dělení vizuální reprezentace umožňující soustředit se na důležité části systému, popřípadě reflektovat na zvolené zájmové části
- Tolerance chyb
  Robustnost rozhraní, jednoduchá možnost návratu do předchozího stavu aplikace
- Zřejmost
  Využití obecně známých konceptů, snaha vyhnout se zbytečně komplikovaným řešením, redukce komplexity

Při úspěšném dodržení výše zmíněných principů značně zvýšíme kvalitu výsledné aplikace, což může mít za důsledek mimo jiné i rychlejší nalezení řešení v rámci sledované problematiky (rychlejší nalezení vzorů v trendech, rychlejší odhalení anomálií v zatížení atd.). Z tohoto směru je tedy snaha o vytvoření kvalitního uživatelského rozhraní pro úspěch celé aplikace velmi důležitá.

2.2.1 Základní struktura vizualizační aplikace

S návrhem uživatelského rozhraní úzce souvisí zvolená základní struktura vizualizační aplikace. Ta se bude primárně odvíjet od následujícího diagramu, jež znázorňuje základní strukturu rozestavění aplikace:

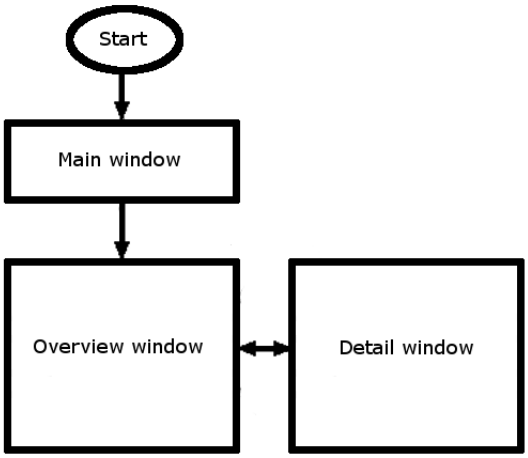


Obrázek 2.2: Diagram návrhu struktury vizualizační aplikace se statickými detaily

Jak lze z obrázku vyčíst, tato varianta návrhu počítá s možností vytvoření několika nezávislých detailních pohledů. V rámci hlavního okna (Main window), si uživatel zvolí, jakou část zatížení chce sledovat (procesor, paměť) a v jakém časovém horizontu (od, do). Na základě tohoto výběru se mu poté zobrazí přehledové okno (Overview window), jež sleduje celkové zatížení jednotlivých serverů (supervizorů) ve zvoleném intervalu a to za využití vizualizační techniky heatmap. V tomto okně poté uživatel může dále selektovat specifický server a časový interval s potenciálně nižší granularitou. Na základě této selekce se mu poté zobrazí detail (Detail window), ve kterém je vyobrazen rozpad zatížení konkrétního serveru na zatížení jednotlivých VM prostřednictvím složeného plošného, potažmo sloupcového grafu. V případě této varianty pak tedy může uživatel takovýchto detailních pohledů vytvořit více, přičemž tyto vzniknou opětovnou selekcí parametrů v přehledovém okně. Více detailních pohledů pak potenciálně může zjednodušit orientaci v dané problematice.

Na základě implementovaného prototypu (viz. níže) se ovšem ukázal nedostatek předchozí varianty v podobě nízké propojenosti přehledového a detailního pohledu (vzájemné propojení je v tomto případě žádoucí, viz. kapitola Analýza 1), kdy změny v jednom se neprojevují v druhém a naopak, právě

vlivem udržení možnosti většího množství detailních pohledů (dynamické přizpůsobování zcela neguje výhodu většího počtu detailních pohledů, všechny pohledy při změně začnou zobrazovat totéž). Alternativou je tedy přístup s jedním přehledovým a jedním detailním pohledem, které jsou však dynamicky aktualizovány v závislosti na provedených změnách. Schéma tedy poté vypadá následovně:



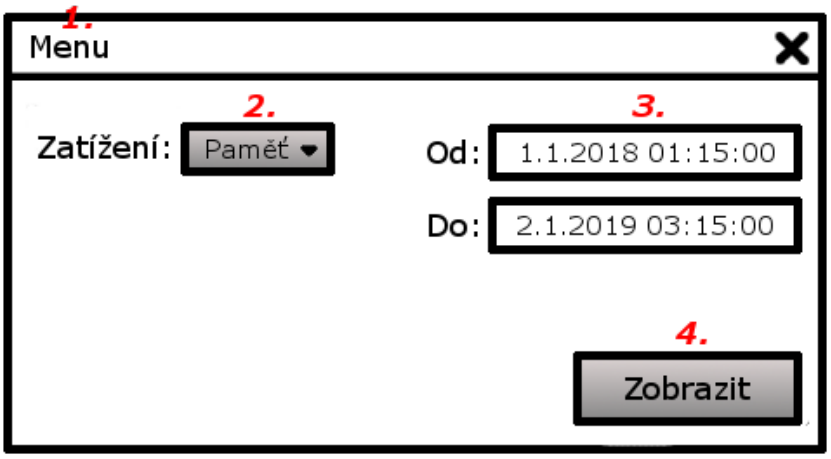
Obrázek 2.3: Diagram návrhu struktury vizualizační aplikace s dynamickým detailem

I tento přístup má však své nevýhody, předně tedy ztrátu schopnosti porovnávat více detailů naráz. Z tohoto důvodu bylo jako explicitní řešení zvolen přístup s dynamickým pohledem, přičemž uživateli bude dána v rámci hlavního okna možnost tuto variantu změnit na přístup s více statickými detaily. Obě varianty tedy budou v rámci výsledné aplikace přítomny.

2.2.2 Wireframy

Nyní se podíváme na jednotlivé návrhy rozhraní konkrétních oken vizualizační aplikace, jež jsou znázorněny pomocí wireframů.

2.2.2.1 Main window



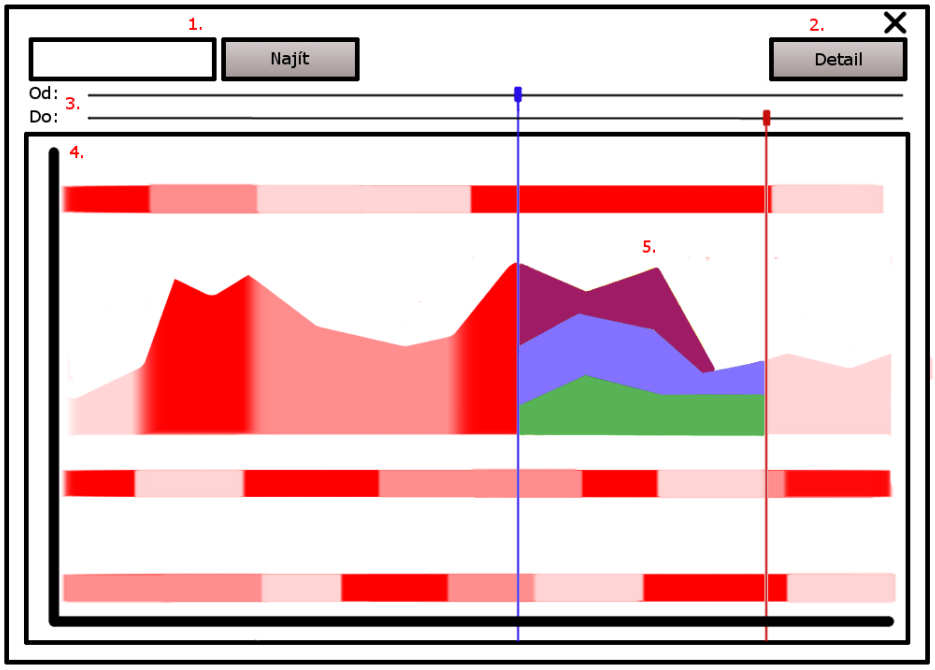
Obrázek 2.4: Wireframe hlavního okna

Hlavní okno představuje výchozí pozici, jež je zobrazena uživateli po spuštění aplikace. Jak je z wireframu vidět, snahou je ponechat úvodní okno jednoduché a stručné. Předně v tomto okně uživatel volí typ zatížení, které chce prostřednictvím vizualizace sledovat, a to skrze volbu v rámci combo boxu (*2.*). Logicky jsou zde tedy dvě možnosti výběru právě mezi paměťovým a procesorovým zatížením, do budoucna pak potenciálně i jiným (vytížení internetové komunikace cloudu apod.).

Další důležitou částí je výběr časového intervalu , kdy uživatel zadá do polí Od a Do (*3.*) přesný časový údaj, jež interval jednoznačně vymezí. Na základě těchto dvou parametrů (typ a interval) je poté aplikací vyslán požadavek na backend, kde dojde k agregaci dat z databáze a takto redukovaná data se poté zašlou zpět do hlavní části pro vizualizaci. Po stisknutí tlačítka Zobrazit (*4.*) se následně objeví přehledového okno (Overview window) dle zvolených parametrů.

Součástí hlavního okna je i menu v horní části (*1.*), v rámci kterého může uživatel mimo jiné zobrazit nápovědu nebo změnit některé parametry nastavení aplikace, například zmiňované zobrazení jednoho dynamického či více statických detailů, viz. výše.

2.2.2.2 Overview window



Obrázek 2.5: Wireframe přehledového okna

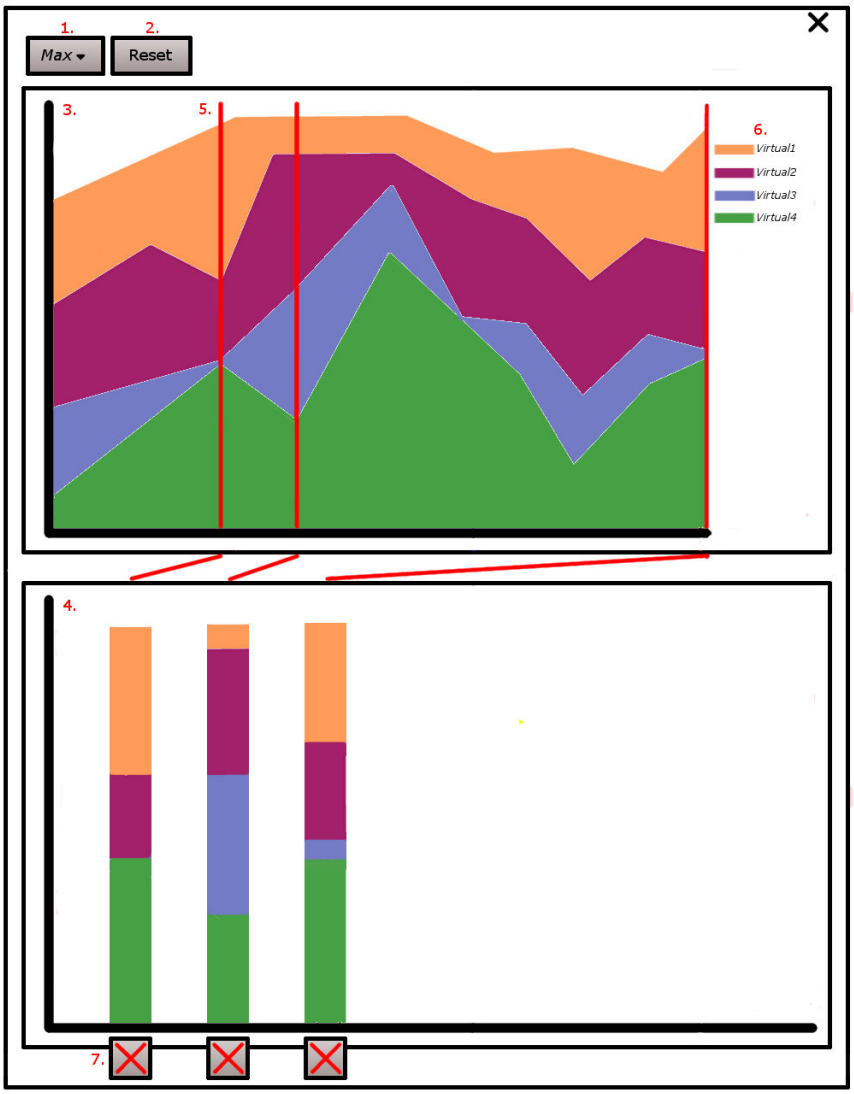
V rámci tohoto okna se dostáváme již k první vizualizaci získaných dat. V tomto případě se jedná o zobrazení celkového zatížení jednotlivých serverů přes vybraný časový interval. Pomocí techniky heatmapy může uživatel v grafu (*4.*) velmi rychle zjistit, kdy a který server byl více či méně vytížen a potenciálně tak včas zjistit vznikající problémy. V rámci tohoto okna má rovněž uživatel k dispozici šoupátka (slidery) Od a Do (*3.*), pomocí nichž může ohraničit podinterval v grafu níže, přičemž poté, co si tento interval zvolí a následně i zvolí příslušný server (přímá interakce s grafem - např. kliknutí myši), se zobrazí metodou Focus and Context preview detailního pohledu (*5.*). Tento detail představuje rozložení zátěže mezi konkrétní virtuály v rámci zvoleného podintervalu, načež zbytek heatmapy (mimo zvolený interval) je poté přetransformován do plošného grafu bez rozpadu na intervaly (ponechány původní barvy heatmapy). Díky tomu může uživatel rychleji analyzovat co se v rámci cloudu děje, aniž by byl nucen otevírat detailní okno (viz. níže) a zároveň může tyto informace porovnávat v rámci kontextu okolí.

Další důležitou částí je možnost vyhledat konkrétní VM prostřednictvím kolonky (*1.*) v horní levé části okna, kde uživatel zadá přesné jméno hledaného virtuálu. Po kliknutí na tlačítko Najít se poté, pokud VM v daném intervalu na nějakém serveru existuje, zobrazí preview daného serveru na kterém vir-

tuál existuje, a to přes celý časový interval, načež hledaný VM je barevně zvýrazněn. Pokud virtuál v daném časovém intervalu neexistuje, je uživateli zobrazena prostřednictvím dialogového okna chybová hláška.

Poslední částí okna je poté tlačítko Detail (*2.*) jehož prostřednictvím uživatel otevře detailní okno (Detail window) poté, co si zvolí konkrétní server a časový podinterval pomocí sliderů. Díky tomu může dále hlouběji analyzovat dění na konkrétním serveru.

2.2.2.3 Detail window



Obrázek 2.6: Wireframe okna detailu

Detailní okno, jak již název napovídá, slouží především k analýze detailního pohledu nad konkrétním serverem prostřednictvím vizualizace zatížení rozloženého mezi jednotlivé VM, a to primárně prostřednictvím techniky složeného plošného grafu (*3.*). V tomto grafu jsou tedy jednotlivé virtuály reprezentovány jakožto plochy jež jsou jednoznačně barevně rozlišeny, přičemž implicitní seřazení je odspoda nahoru, kde nejspodnější plocha představuje VM s největším celkovým zatížením přes celý časový podinterval detailu. Toto řazení poté může uživatel měnit prostřednictvím interakce. Ke grafu je rovněž přidružena legenda (*6.*), jež logicky přiřazuje k jednotlivým barvám jméno daného VM.

Další vlastností tohoto grafu je následně možnost zvolení si konkrétního časového okamžiku za účelem detailnější analýzy. Volba probíhá přímou interakcí s oknem (kliknutí myši), je znázorněna vertikální červenou úsečkou (*5.*) a zapříčiní vytvoření složeného sloupce v rámci dolního grafu (*4.*), jež slouží právě k lepší vizualizaci konkrétního okamžiku a umožňuje lepší sledování poměrů zatížení jednotlivých VM. Jednotlivé sloupce jsou poté propojeny s okamžiky v horním grafu, opět pomocí červené úsečky. Při výběru časového okamžiku rovněž vznikne tlačítko pro smazání daného detailu (*7.*), a to pod příslušným složeným sloupcem. Toto tlačítko po stisknutí pak daný detail odebere.

Dále je součástí detailního okna i combo box (*1.*) pro změnu zarovnání VM, tedy zarovnání zdola nahoru podle maxima (implicitní volba při vytvoření detailního pohledu) či minima (roz. maximální a minimální zatížení přes daný interval). V neposlední řadě je zde rovněž tlačítko Reset (*1.*), které smaže všechny vybrané okamžiky (dolní složený sloupcový graf je prázdný) a uvede horní složený plošný graf do stavu, ve kterém se nacházel při otevření detailního okna (VM zarovnané podle max zdola nahoru).

Na závěr, jak již bylo zmíněno výše, dynamičnost či statičnost detailu je určena volbou uživatele, přičemž pokud je detailní okno dynamické, veškeré změny z přehledového okna se sem propagují, jinak nikoliv.

2.3 Základní algoritmy

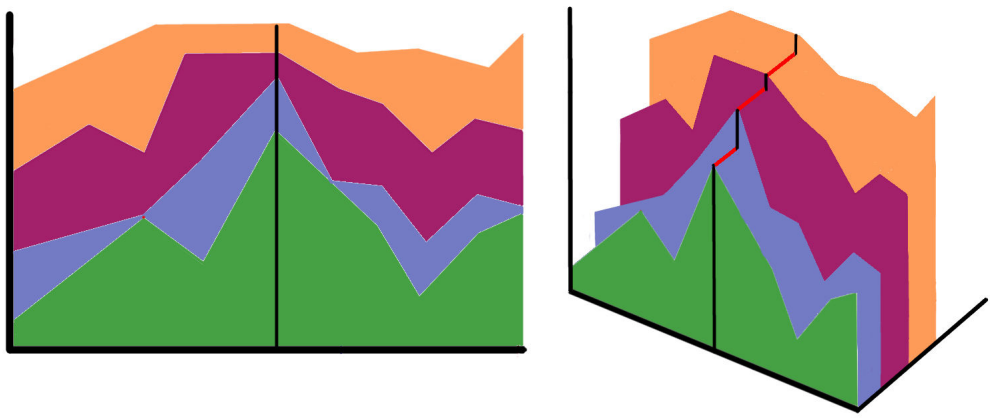
V této kapitole jsou popsány některé ze základních algoritmů, které je v rámci vytvoření výsledné aplikace třeba implementovat. Některé z těchto algoritmů pak úzce souvisí s volbou technologií (roz. nutnost jejich implementace se vztahuje k vybranému technologickému řešení - v případě jiných technologií takovéto řešení může být významně pozměněno).

2.3.1 Data pro složený plošný graf

První algoritmus úzce souvisí s knihovnou QCustomPlot, kterou budeme používat při vykreslování jednotlivých grafů (viz. kapitola Implementace 3). Tato knihovna totiž neumí přímo vykreslovat složený plošný graf (jednotlivé plochy začínají tam, kde končí plocha pod nimi v daném bodě osy X), místo toho

však umožňuje použití klasického plošného grafu - jednotlivé plochy začínají všechny v počátku osy Y a vzájemně se překrývají.

Díky této vlastnosti jsme schopni po úpravě dat tento plošný graf simulovat, a to následujícím způsobem: V prvé řadě je důležité zjistit pořadí, ve kterém budou jednotlivé plochy (VM) seřazeny. Následně vezmeme v úvahu fakt, že aby byla plocha zepředu zakryta plochou jinou (barva ploch musí být netransparentní), musí tato být vykreslena dříve, tedy budeme postupovat odzadu do předu. To znamená, že první se bude vykreslovat plocha nejvíce nahoře. Hodnota v jednotlivých bodech této plochy je poté součtem všech hodnot ploch (zatížení virtuálů), které se nachází pod ní. Jinými slovy, jediným VM, pro který nedojde k úpravě hodnot, je ten, který je ve vizualizaci nejvíce dole, což znamená že má celkově přes daný interval největší zatížení a je tedy zarovnán k ose X. Tento je taktéž vykreslen jako poslední, neboť je nejvíce "vpředu". Základní princip úpravy je znázorněn na následujícím obrázku:



Obrázek 2.7: Složený plošný graf z pohledu uživatele (vlevo) a ilustrace vnitřní reprezentace pomocí obecných plošných grafů (vpravo)

Je třeba podotknout, že k této úpravě dochází pochopitelně pouze v rámci příslušné instance vizualizace, tedy základní data zůstávají nezměněna, i kvůli možnosti interakcí přeskupovat pořadí jednotlivých ploch. Úpravu dat pro jednotlivé virtuály tedy můžeme poté vyjádřit následujícím pseudokódem:

Algorithm 1 Stacked area chart's data preparation algorithm

```
1: function PrepareStackedAreaChartData(data, n)
2:     Let sum[1...n] be new array of adjusted data
3:     for i = n to 1 do
4:         sum[i] = data[i]
5:         for j = 1 to i do
6:             sum[i] = sum[i] + data[j]
7:         end for
8:         Add area to graph with sum[i] as data specifying top boundary
9:     end for
10: end function
```

Písmeno **n** v tomto případě představuje počet VM, přičemž data jsou reprezentovány mapou, kde klíč je časový údaj a hodnota příslušné vytížení (paměť či procesor, dle předchozí volby). V tomto případě se však nemusíme omezovat k přistupování pouze na určitý časový okamžik, avšak tato data bereme přes celý vybraný interval (na začátku z mapy vyselektujeme relevantní časové okamžiky). Data[j] tedy představují údaje o zatížení v rámci celého vybraného intervalu pro virtuální stroj **j**.

Algoritmus rovněž předpokládá, že data, respektive jednotlivé VM, jsou již seřazeny od nejméně po nejvíce zatížený (obráceně pokud je zvoleno řazení min v rámci combo boxu). Toto řazení provedeme jednoduše na základě součtu zatížení pro každý konkrétní VM zvlášť (přes zvolený interval) v rámci inicializace detailního okna, případně při změně typu řazení (min,max).

2.3.2 Data pro přehledovou heatmapu

Technika heatmapy, podobně jako složený plošný graf, není implicitní součástí knihovny QCustomPlot, kterou budeme v rámci implementace jednotlivých technik používat. Heatmapa se však dá poměrně elegantně simulovat za pomocí horizontálních složených sloupcových grafů, přičemž jeden složený sloupcový graf simuluje řádek heatmapy, tedy konkrétní server. Jednotlivé dílčí části tohoto složeného sloupce poté představují zatížení v rámci konkrétního časového intervalu, což koresponduje se sloupci heatmapy. Barva na škále od bílé po sytě červenou poté určuje míru zatížení, přičemž je kladen důraz na pop up efekt v případě vysokého zatížení.

V prvé řadě je tedy potřeba pro každý server a každý zobrazený timestamp spočítat celkové zatížení serveru v závislosti na zatížení jednotlivých VM. To lze vyjádřit jednoduše následujícím algoritmem:

Algorithm 2 Heatmap's data preparation algorithm

```
1: function PrepareHeatmapData(servers)
2:     for i = 1 to servers.size do
3:         Let sU[start...end] be map of this server's total usage
4:         for j = start to end do
5:             for k = 1 to servers[i].VM.size do
6:                 sU[j] = sU[j] + servers[i].VM[k].data[j]
7:             end for
8:             Add bar to graph on row i, column j and colour based on sU[j]
9:         end for
10:    end for
11: end function
```

Kde **start** a **end** značí ohraničení časového intervalu (krajní timestampy), pro který je přehled vizualizován. Pole servers pak obsahuje jednotlivé servery, jež dále obsahují pole vlastnících VM, které v sobě dále mají jednotlivá data zatížení v korespondenci s příslušnými okamžiky. V případě, že zatížení jednotlivých VM je relativní k virtuální paměti/procesoru a nikoliv k paměti/procesoru serveru, je třeba tato přepočítat, a to jednoduše následujícím způsobem v závislosti na poměru mezi pamětí/procesorem serveru a virtuální pamětí/procesorem příslušného VM:

Algorithm 3 Recalculate usage data relative to server

```
1: function RelativeUsage(usage, maxVM, maxServer)
2:     return (maxVM * usage) / maxServer
3: end function
```

Toto přepočítání pak bude probíhat ještě před ukládáním do databáze v rámci zpracovávání dat.

2.3.3 Umístění a šířka řádků heatmapy

S přehledovou heatmapou rovněž úzce souvisí i možnost zobrazení si preview v rámci tohoto grafu. V takovém případě musí dojít ke zvětšení vybraného serveru, respektive tedy šířky příslušného řádku heatmapy (horizontálního složeného sloupce), a zmenšení ostatních, a to tak, aby byl prostor vyhrazený pro heatmapu nezměněn, avšak zároveň byl dán větší prostor právě pro zobrazení předběžného náhledu. Umístění jednotlivých sloupců (tj. souřadnice pozice i z algoritmu 2) se poté rovněž mění v závislosti právě na selekci serveru, stejně tak jako zmíněná šířka sloupců. Vše poté vztahujeme relativně k výšce celého grafu (osa Y) tak, aby poměry byly zachovány i při změně rozlišení okna. V případě výpočtu pozice řádků heatmapy lze použít následující algoritmus:

Algorithm 4 Heatmap's row position calculation algorithm

```
1: function CalculateRowsPosition(rows)
2:     Let maxY be total height of Y axis
3:     for i = 1 to rows.size do
4:         if no server selected then rows[i].position = (i/rows.size) * maxY
5:         else
6:             if i < selected server then
               rows[i].position = 0.5 * (i/rows.size) * maxY
7:             else
8:                 if i > selected server then
                   rows[i].position = 0.5 * (i/rows.size) * maxY + maxY/2
9:                 else
                   rows[i].position = 0.5 * (i/rows.size) * maxY + maxY/4
10:                end if
11:            end if
12:        end if
13:    end for
14: end function
```

Pokud není žádný server vybrán, jsou jednotlivé řádky rovnoměrně rozprostřeny po ose Y. Pokud je nějaký vybrán, zabírá více místa a tedy je třeba, aby byl od ostatních více vzdálen, načež zbylé se k sobě přiblíží (analogicky, šířka nevybraných řádků se zmenší, šířka vybraného řádku se zvětší). Z tohoto důvodu je tento posunut o jednu čtvrtinu celkové viditelné délky osy Y. Řádky nad ním se poté dále posunou, tentokrát o celou polovinu (pozice je střed řádku, tedy horní polovina vybraného řádku je další jedna čtvrtina celkové délky osy Y). V případě výpočtu šířky sloupců, jež reprezentují řádky heatmapy, je algoritmus analogický. Místo pozice se zde však počítá přímo s pixely, jež reprezentují určitý poměr osy Y. Tyto jsou poté přiřazeny jako výsledná šířka daného složeného sloupce. Na závěr je vhodné dodat, že transformace pozic jednotlivých řádků a změna jejich šířky při selekci konkrétního serveru bude animována, viz. níže.

2.3.4 Náhledové okno a okolní kontext

Náhled v rámci přehledového grafu a kontextem tohoto náhledu, jenž reprezentuje transformaci řádku heatmapy do obrysu složeného plošného grafu, budou vnitřně reprezentovány jako vlastní okna, přičemž vlastní náhled se bude vykreslovat po kontextu, tedy bude ho překrývat (kontext se tvoří pro celý interval, včetně časového podintervalu pro náhled a detail). V tomto případě je důležité tyto okna umístit na správná místa do přehledového grafu. Rovněž je podstatné, aby vzniklá okna měla správnou šířku a výšku a aby neobsahovala nic jiného, než vlastní grafy (tedy ani osy, mřížky, legendy atd.).

Za účelem správného umístění oken musíme v pixelech spočítat jednak

šířku a výšku okna, tak i bod (opět v pixelových souřadnicích přehledového okna), do nějž bude umístěn levý dolní roh náhledového okna. X souřadnice tohoto bodu je pro případ náhledu určena jako levá hranice časového podintevalu (from), v případě kontextu náhledu je to levá hranice celého časového intervalu (počátek osy X). Y souřadnice je poté pro obě okna stejná, tedy dolní hranice rozšířeného řádku heatmapy pro server, jenž je vybrán. Pozice obou těchto oken se poté musí přepočítat pokaždé, když dojde ke změně rozlišení přehledového okna (resize event) kvůli rovnoměrnému zvětšování/zmenšování šířky řádku heatmapy a tedy i výšky oken a Y souřadnice bodu, do nějž okna umisťujeme. Právě pro přepočet těchto hodnot ze souřadnic v grafu, jenž známe, do pixelových souřadnic okna, musíme použít následující algoritmus:

Algorithm 5 Graph coordinates to pixel coordinates transformation algorithm

```
1: function GraphCoordToPixel(value)
2:     return ((value - min) / (max - min)) * (maxPix - minPix) +
           minPix
3: end function
```

Kde min a max jsou minima a maxima příslušné souřadné osy v daném okně a minPix a maxPix jsou pixelové souřadnice minima a maxima daného okna, přičemž pixelové souřadnice rohů (min a max v osách) daného okna (či widgetu vůči hlavnímu oknu) implicitně známe.

Co se týče vlastních grafů, je jejich vytvoření podobné, až identické s vytvořením složeného plošného grafu v rámci detailu (viz. algoritmus 1). K jediné změně dochází v případě kontextu náhledu, kde zde nemáme plochy pro jednotlivé VM, jako v případě detailu, ale naopak plochy pro jednotlivé základní časové intervaly (interval mezi dvěma nejbližšími timestampy). Tyto jsou poté barevně rozlišeny obdobně jako vlastní části heatmapy, tedy v závislosti na celkovém zatížení daného serveru pro tento interval. Upravený algoritmus 1 v tomto případě tedy vypadá následovně:

Algorithm 6 Preview context data preparation algorithm

```
1: function PreparePreviewContextData(data, n)
2:     Let sum[start..end] be new array of adjusted data
3:     for i = start to end do
4:         sum[i] = 0
5:         for j = 1 to n do
6:             sum[i] = sum[i] + data[j].timestamp[i]
7:         end for
8:     end for
9:     for i = start + 1 to end do
10:        Add area to graph with i as right boundary and i-1 as left boundary
11:        Top boundary is based on sum[i] and sum[i − 1] data
12:        Colour of the area is based on sum[i] data
13:    end for
14: end function
```

Kde **n** představuje opět počet VM daného serveru. Všimněme si, že jelikož zde již počítáme součet zatížení v závislosti na konkrétním časovém intervalu (jednotlivé plochy představují časový interval, nikoliv VM jako v případě detailního pohledu), je třeba z dat o zatížení VM přistupovat pouze k těm, jenž jsou relevantní pro daný časový okamžik.

2.3.5 Animace

Animace jako taková je velmi žádoucím prvkem pro informovanost uživatele o tom, co se v rámci vizualizace děje. Z tohoto důvodu je vhodné, zejména ve vztahu k interakci s vizualizací, animaci zahrnout jakožto součást našeho řešení. Z hlediska implementace nebude k vytvoření animace použita žádná knihovna, je tedy zapotřebí veškerou logiku spojenou s ní naprogramovat. V tomto směru budeme vycházet ze standardního chápání animace jako sledu dílčích obrázků (framů) přes omezený časový interval (circa sekunda pro celou animaci). Bude tedy vytvořena speciální třída, jejíž cílem bude připravit vstupní data (počáteční a koncový frame) a následně tato interpolovat pro každý individuální frame zvlášť. Výsledek poté bude za pomocí časovače vykreslen, čímž vznikne postupná animace.

V případě naší aplikace budeme celkem animovat tři činnosti: Změnu selekce serveru v rámci přehledu pro možnost zobrazení náhledu, selekci prioritního VM v rámci složeného plošného grafu respektive složeného sloupcového grafu detailu a změnu řazení (min/max) jednotlivých VM v rámci vizualizace detailu (složený plošný i složený sloupcový graf). Na selekci VM, potažmo změnu řazení budou současně reagovat oba dva grafy v rámci detailu, tedy budeme oba animovat paralelně.

V prvé řadě si při vytváření animace v závislosti na typu činnosti a obdržených datech spočítáme počáteční a koncová data, tedy data která vizu-

alizovaná budou představovat první a poslední frame animace, přičemž tato příprava dat bude pravděpodobně pro jednotlivé činnosti různá. Interpolace jednotlivých framů pak bude probíhat za využití následujícího algoritmu:

Algorithm 7 Interpolate animation data algorithm

```
1: function INTERPOLATE(startdata, enddata, frames, n)
2:     Let interpolated[1..frames][1..n] be new 2D array of interpolated data
3:     for i = 1 to frames do
4:         for j = 1 to n do
5:             interpolated[i][j] = startdata[j] + i * (1/nframes) *
               (enddata[j] − startdata[j])
6:         end for
7:     end for
8: end function
```

Takto interpolovaná data pro jednotlivé framy budou poté předána zpět k příslušným grafům, přičemž následně proběhne animování. To v zásadě znamená, že se spustí časovač, který v pravidelných intervalech překreslí celý graf interpolovanými daty pro daný frame, načež pořadí framu inkrementuje, a to až do předem definovaného konečného počtu, kdy animace skončí.

2.3.6 Selekce VM a časového okamžiku

Důležitou součástí vizualizační aplikace je pochopitelně i možnost interagovat s příslušnými grafy a přizpůsobovat si je podle potřeby. Předně se tedy jedná o selekci časového intervalu a serveru v rámci přehledu, načež se zobrazí náhled a potažmo je možné si zobrazit detailní pohled. V rámci tohoto detailu je poté důležité mít možnost přeskupit pořadí vizualizovaných VM, a to prostřednictvím přesunu konkrétní plochy dolů k zarovnání k ose X, načež ostatní plochy (VM) se příslušně přerovnají (resp. jejich hodnoty). Jelikož v rámci implementace (roz. knihovny QCustomPlot) není k dispozici žádný údaj, na základě kterých by bylo možné jednoduše určit na kterou plochu (VM) uživatel kliknul, je potřeba pomocí postupné interpolace toto místo explicitně určit.

Za tímto účelem je třeba naimplementovat pro selekci VM v rámci složeného plošného grafu následující algoritmus:

Algorithm 8 Select VM algorithm for stacked area chart

```
1: function selectVirtual(data, x, y, n)
2:     sumLeft = 0
3:     sumRight = 0
4:     for i = start to end − 1 do
5:         if x ≥ i and x < i + 1 then
6:             left = i
7:             right = i + 1
8:         end if
9:     end for
10:    for j = 1 to n do
11:        sumLeft = sumLeft + data[left]
12:        sumRight = sumRight + data[right]
13:        usage = sumLeft + (sumRight − sumLeft) ∗
                   ((x − left)/(right − left))
14:        if y ≤ usage then return j
15:        end if
16:    end for
17: end function
```

Algoritmus předpokládá, že jak X souřadnice, tak Y jsou již relativní vůči osám daného grafu. Jak lze z pseudokódu vyčíst, nejprve je třeba v závislosti na X souřadnici bodu nalézt levý a pravý nejbližší časový okamžik, pro který máme k dispozici data. Následně postupujeme po jednotlivých plochách (VM) zdola nahoru (předpokládá se že data už jsou seřazena, jinak bychom nemohli mít graf), přičemž průběžně interpolujeme zatížení v závislosti na X souřadnici bodu a datech pro levý a pravý časový okamžik. Kvůli podstatě implementace složeného plošného grafu pak celkové interpolované zatížení sčítáme postupně s předchozími vypočítanými hodnotami. V každé iteraci pak zkontrolujeme, zdali Y souřadnice je menší než interpolovaná hodnota. Jelikož postupujeme zdola nahoru, v momentě, kdy je tato podmínka poprvé splněna, je jednoznačně určeno, na kterou plochu v rámci grafu uživatel kliknul. V případě selekce VM pro složený sloupcový graf je algoritmus obdobný, s tou výjimkou, že nemusíme zatížení interpolovat (pouze nasčítáváme zatížení konkrétních VM) a podle souřadnice X nehledáme nejbližší časové okamžiky, nýbrž konkrétní složený sloupcový graf.

V případě selekce timestampu nebo konkrétního serveru v rámci overview pak stačí pouze najít nejbližší bod na ose X (ose Y při selekci serveru) k bodu, na který uživatel kliknul. V tomto směru tedy postačí následující jednoduchý algoritmus:

Algorithm 9 Select timestamp algorithm

```
1: function findNearest(x)
2:     minDist = +∞
3:     for i = start to end do
4:         dist = |x − i|
5:         if dist < minDist then
6:             minDist = dist
7:             nearestPoint = i
8:         end if
9:     end forreturn nearestPoint
10: end function
```

Kde **start** a **end** značí počáteční, respektive koncový časový okamžik, pro který máme data. Alternativně se pak může jednat o první a poslední server v případě selekce serveru pro přehledový graf.

2.3.7 Agregace dat

Agregace dat je důležitým prostředkem k tomu, jak zpřehlednit vizualizaci a celkově usnadnit manipulaci a interakci s aplikací. Díky potenciálně obrovskému množství dat by bylo v případě větších vybraných časových intervalů (měsíc, rok) velmi náročné všechna tato data efektivně vizualizovat, respektive by mohlo dojít k významnému zpomalení, přičemž některé další prvky aplikace, jakými jsou třeba jednotlivé animace přechodů, by byly značně negativně poznamenány (animace vždy interpoluje data nanovo dle selekce, interpolace velkého množství dat je značně náročná). Díky tomu je daleko jednodušší data nejprve agregovat do několika málo časových bodů (timestampů), a to na základě rozsahu vybraného časového intervalu. Takto agregovaná data lze následně jednodušeji vizualizovat, potažmo s nimi provádět další operace (interpolace v animaci atd.). Agregace tedy probíhá dle následujícího algoritmu:

Algorithm 10 Aggregate data algorithm

```
1: function AGGREGATE(data, from, to, n)
2:     Let aggregated[1..n] be new array of aggregated data
3:     dist = (to − from)/n
4:     for i = 1 to n do
5:         agg = 0
6:         count = 0
7:         lower = data.lowerBound((start + i ∗ dist) − dist/2)
8:         upper = data.upperBound((start + i ∗ dist) + dist/2)
9:         for j = lower to upper do
10:            agg = agg + data[j]
11:            count = count + 1
12:        end for
13:        aggregated[i]= agg / count
14:    end for
15: end function
```

Kde **n** je počet agregátů (timestampů), do kterých data agregujeme, lowerBound a upperBound pak zvolí nejbližší menší, respektive větší klíč v mapě dat k zadané hodnotě. V případě, že pro daný interval existuje menší počet reálných časových dat (timestampů) než je daný počet n, je agregace pochopitelně zbytečná. Smysl pak dává maximálně upravit (rozšířit či zúžit) zvolený časový rozsah tak, aby krajní hodnoty náležely reálným časovým datům (timestampům).

2.4 Prototyp

Jak již bylo zmíněno, v rámci fáze návrhu byl vytvořen i jednoduchý funkční prototyp, jehož primárním cílem bylo odhalit potenciální nedostatky a obecně vyhodnotit kvalitu zvolených vizualizačních technik pro dílčí uživatelské úkoly. V této kapitole krátce shrneme zjištěné poznatky, které poté dále ovlivnily implementaci finální vizualizační aplikace.

2.4.1 Zvolené techniky pro testování v rámci prototypu

Hlavním cílem prototypu bylo primárně ověřit použitelnost zvolených vizualizačních technik a ukázat, kde jsou jejich nedostatky ve vztahu k sledovaným cílům. Na základě výstupu z analýzy byly pro ověření funkčnosti vybrány techniky složeného plošného grafu pro vizualizaci zatížení virtuálních strojů konkrétního serveru přes specifikovaný časový interval, dále složeného sloupcového grafu pro detail konkrétního časového okamžiku v rámci složeného plošného grafu (pro eliminaci horší rozpoznatelnosti poměrů při výrazných rozdílech v zatíženosti) a heatmapy pro přímé porovnávání celkové zatíže-

nosti mezi servery samotnými a pro rychlou detekci abnormalit, respektive výrazného zatížení. Mimo zvolené techniky pak bylo rovněž testováno potenciální možné uspořádání aplikace jako takové, respektive forma vztahu mezi jednotlivými pohledy. V rámci prototypu nebyly explicitně implementovány techniky pro zlepšení interakce ani výhody přístupu Focus and Context (náhled v rámci přehledu). Došlo tedy nepřímo i k testování vlivu absence těchto prvků na celkovou použitelnost a uživatelskou přívětivost aplikace.

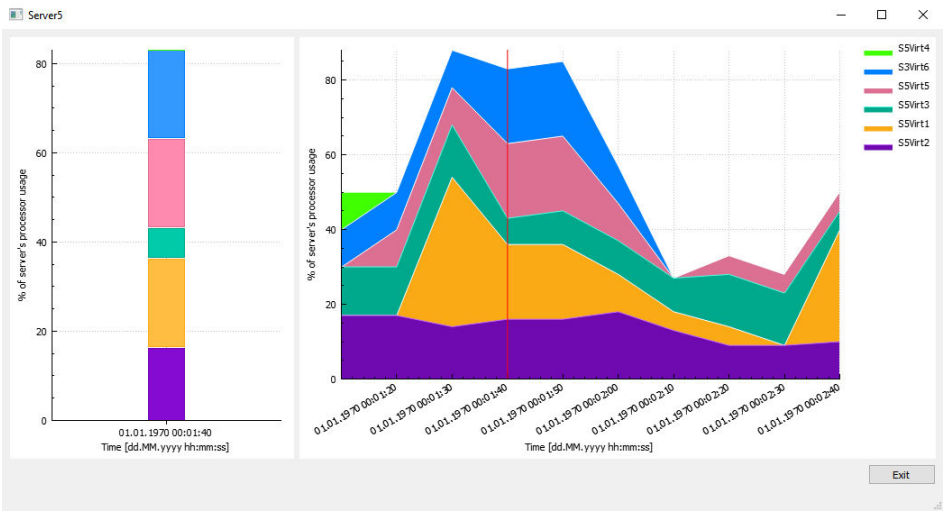
2.4.2 Struktura prototypu

Na základě vybraného návrhu byla testovaná hierarchie mezi zobrazením serverů a VM rozdělena do dvou pohledů - přehledové okno a detailní okno. Přehledové okno obsahovalo vizualizaci celkového zatížení serverů pomocí techniky heatmapy, kde osa X je osou časovou, na ose Y jsou poté jednotlivé hostující servery. Odstíny červené zde reprezentují celkové zatížení daného serveru (bílá nízké zatížení, syté červená vysoké). Alternativou by mohla být barevná škála modrá-zelená-žlutá-červená, ta však výrazně omezuje využití barev v přidruženém detailním pohledu pro kategorizaci jednotlivých VM. Právě z důvodu potenciálního matení uživatele barevnou podobností kategorií nebo objektů, které spolu nijak nesouvisí (% vytížení serveru - konkrétní virtuální stroj), byla prioritně zvolena jednobarevná škála (roz. jeden barevný odstín - červená) s měnící se intenzitou. Na následujícím obrázku můžeme pozorovat celkový pohled na přehledové okno prototypu:

Obrázek 2.8: Přehled CPU vytíženosti serverů v rámci prototypu

Detailní pohled byl následně představován oknem se dvěma grafy: V prvé řadě se jednalo o složený plošný graf, kde každá plocha zobrazuje jeden virtuální stroj hostujícího serveru, který v daném časovém intervalu existoval/existuje. Plochy jsou rovněž vybarveny rozdílnými barvami pro lepší identifikaci a vpravo od grafu je za tímto účelem přítomna legenda. Jednotlivé virtuální stroje jsou taktéž v grafu seřazeny, a to tím způsobem, že VM, který přes daný interval nejvíce vytěžoval hostující server, je první nad osou X (časovou osou). Ostatní se poté obdobně řadí zdola nahoru - tedy následně druhý nejvíce vytěžující, třetí, atd.

Druhým grafem, který je zprvu prázdný, byl složený sloupcový graf. Ten slouží především pro zobrazení poměrů mezi jednotlivými VM pro konkrétní časový okamžik a jeho zobrazení je tedy podmíněno vybráním tohoto okamžiku v plošném grafu. Při selekci rovněž dochází k interpolaci, jinými slovy uživatel nemusí přesně "trefit" okamžik v grafu, vybere se vždy ten nejbližší místu, kam kliknul. Na následujícím obrázku můžeme vidět konkrétní detailní pohled pro daný server ve vybraném časovém intervalu:



Obrázek 2.9: Detailní pohled na vytíženost jednotlivých virtuálních strojů v rámci prototypu

2.4.3 Výsledky

Mezi nejzásadnější nedostatky zvolených technik, které prototyp odhalil, patří problém s barevným rozlišením jednotlivých VM v rámci složeného plošného grafu, potažmo tedy složeného sloupcového grafu. Obecně může platit, že virtuálních strojů může být několik desítek v rámci jednoho serveru. V takovém případě se ukázalo, že je velmi obtížně až nemožné zvolit dostatečný počet barevných odstínů pro jejich bezpečné rozlišení. Navíc díky tomu, že přehledová heatmapa využívá odstíny červené, nemůžeme tyto v rámci detailního

pohledu využít (mohlo by dojít k implikaci neexistující vztahu), což dále limituje množství VM, které můžeme barevně odlišit. Jedno z řešení může být obecně zobrazení jen limitovaného počtu virtuálních strojů - uživatel primárně sleduje maxima, respektive minima v rámci daného okamžiku. Seřazením můžeme tedy eliminovat některé méně významné stroje z hlediska zamýšleného pozorování (omezíme se jen na určitý počet vizualizovaných VM). To však značně limituje přesnost zobrazení a v některých případech použití tohoto omezení může komplikovat výslednou vizualizaci (sledování konkrétního VM v rámci kontextu).

V rámci prototypu vyšlo dále najevo i několik nedostatků stran uživatelské interakce. Absence intuitivnějších forem interakce a jejich náhrada za interakci pomocí tlačítek a radio buttonů se ukázala v některých případech jako zbytečně komplikovaná a tedy je ve výsledné aplikaci nahrazena za přímější formu interakce prostřednictvím klikáním a pohybu myši v grafu (Zoom and Pan, Selekce). Rovněž možnost selekce právě jednoho časového okamžiku pro detail (složený sloupcový graf) byla nedostatečná, je tedy zapotřebí mít možnost zobrazení detailů pro více okamžiků. V neposlední řadě se taktéž negativně projevila absence animací a některých provazujících prvků mezi grafy, což zapříčinilo daleko horší orientaci v tom, jaké změny ve vizualizaci na základě uživatelské interakce interakce proběhly a kde se uživatel v dané chvíli nachází.

Vyjma těchto několika nedostatků, které navíc v některých případech souvisely spíše s ranou verzí prototypu, se zvolené vizualizační techniky poměrně osvědčily. Je možné, že s přibývajícími požadavky na finální aplikaci bude třeba tyto techniky doplnit o nějaké další, kupříkladu liniové grafy, avšak současné výsledky tomu nenasvědčují. Daleko pravděpodobnější se jeví vylepšení celkové vizualizace prostřednictvím hlubšího provázání stávajících technik, respektive pohledů, např. prostřednictvím metody Focus and Context. Jinými slovy, pro výslednou vizualizační aplikaci jsou zvolené techniky, jež byly v rámci prototypu testovány, ponechány.

Nepřímou výhodou prototypu pak také bylo, že jeho implementace proběhla na stejné platformě a za použití stejných technologií, které byly následně použity i při tvorbě finální vizualizační aplikace (viz. níže). Díky celkově pozitivnímu výstupu pak tedy výsledný prototyp posloužil jako výchozí bod pro její následnou implementaci.

Kapitola 3

Implementace

Tato kapitola se věnuje, jak již název napovídá, detailnějšímu pohledu na výslednou implementaci vizualizační aplikace, jež proběhla na základě výsledků analýzy a následně i jednotlivých návrhů. Hlavním předmětem je tedy stručný popis jednotlivých tříd a jejich metod, jimi obstarávané funkcionality a celkový popis dílčích částí aplikace a jejich funkcí. Součástí je taktéž popis jednotlivých technologií a jejich verze pro případ navazujících prací třetích stran.

3.1 Výběr technologií

Nyní se detailněji podíváme na původně zvažované a zejména pak vybrané technologie, které byly použity při implementaci hlavní části výsledné vizualizační aplikace (a potažmo i při implementaci prototypu). Předně se jedná o výběr použitého programovacího jazyka. Tento výběr pak pochopitelně úzce souvisel s výběrem platformy, pro kterou je aplikace vyvíjena. Základní výběr tedy proběhl mezi následujícími jazyky, jež umožňují jednoduché vytváření grafických rozhraní a vlastních grafů:

- **Javascript**[29]
  JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape. Zpravidla se používá jako interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová pole) nebo vytvářeny animace a efekty obrázků.
- **Java**[30]
  Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems a představila 23. května 1995. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a

různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE), až po rozsáhlé distribuované systémy pracujících na řadě počítačů rozprostřených po celém světě (platforma Java EE).

- **Python**[32]
  Python je vysokoúrovňový skriptovací programovací jazyk, který v roce 1991 navrhl Guido van Rossum. Nabízí dynamickou kontrolu datových typů a podporuje různá programovací paradigmata, včetně objektově orientovaného, imperativního, procedurálního nebo funkcionálního. Python je vyvíjen jako open source projekt, který zdarma nabízí instalační balíky pro většinu běžných platforem (Unix, MS Windows, macOS, Android).

- **C++**[31]
  C++ je multiparadigmatický programovací jazyk, který vyvinul Bjarne Stroustrup a další v Bellových laboratořích AT&T rozšířením jazyka C. C++ podporuje několik programovacích stylů (paradigmat) jako je procedurální programování, objektově orientované programování a generické programování, není tedy jazykem čistě objektovým. V současné době patří C++ mezi nejrozšířenější programovací jazyky.

Vzhledem k tomu, že jako primární platforma pro vývoj byla zvolena desktopová aplikace (viz. Analýza 1) a zejména pak díky větším zkušenostem s jazykem C++ a s jeho frameworky, byl jako hlavní zvolen právě tento programovací jazyk. C++ je velice rozšířený jazyk a existuje nespočet různých knihoven, které významným způsobem mohou ulehčit práci při vlastní implementaci. Na druhou stranu, jeho rozšiřitelnost zejména v rámci webových technologií je značně omazená. Díky tomu bude do budoucna v případě rozšiřování použitelnosti a dostupnosti aplikace i na další platformy potřeba zvolené technologie a tedy i programovací jazyky revidovat.

Další důležitou volbou stran technologií byl výběr základních frameworků a knihoven, které významně usnadňují výslednou implementaci a umožňují lepší práci s grafickými prvky (roz. např. grafy) a tvorbu GUI. Díky robustnosti jazyku C++ existuje velká řada takovýchto knihoven a frameworků, nejznámější z nich jsou pak následující:

- GTK+
- Qt
- Plotutils
- OpenGL
- OWLNext

- wxWidgets

V tomto případě byla opět volba základního frameworku ovlivněna zejména předchozími zkušenostmi. Z tohoto důvodu byl vybrán framework Qt, a to i přesto, že některé alternativy jsou vcelku robustnější, avšak často nepoměrně komplikovanější a práce s nimi by tak potenciálně zabrala daleko více času, přičemž vlastní výsledek by nemusel být o mnoho lepší. Qt umožňuje interaktivně vytvářet uživatelské rozhraní aplikací a celkově tak významně urychlit vývoj. Na druhou stranu neposkytuje nijak závratné zjednodušení implementace grafů, potažmo obecných vizualizačních technik (obecně je často Qt doplněn o např. OpenGL aby se některé tyto nedostatky kompenzovaly). Z tohoto důvodu byl Qt doplněn o speciální knihovnu QCustomPlot, jež je exkluzivní právě pro tento framework a která umožňuje efektivní tvorbu grafů a manipulaci s nimi. Opět je zde potřeba zmínit, že s postupem času a dalším rozšiřováním aplikace, ať už ze strany funkcionality, nebo ze strany použitelnosti na jiných platformách, bude s největší pravděpodobností potřeba tyto zvolené nástroje dále rozšířit o nové, popřípadě zcela změnit původní volbu.

Nyní se tedy podíváme na tyto dva základní nástroje o trochu detailněji.

3.1.1 Qt

Qt [40] je multiplatformní aplikační rámec vytvořený v roce 1999 společností Trolltech, který prostřednictvím nejrůznějších nástrojů umožňuje velmi jednoduchou tvorbu aplikačního softwaru s GUI. Qt navíc dokáže pracovat na nejrůznějších hardwarových i softwarových platformách, a to aniž by došlo k výraznější změně základní kódové struktury, což umožňuje poměrně vysokou míru přenositelnosti výsledné aplikace. Přestože Qt je primárně frameworkem pro jazyk C++, existují i verze pro Python, C, C#, Pascal či Javu. Mimo jiné pak podporuje i použití SQL, správu vláken, zpracování (parsování) XML a JSON či práci s obecnou grafikou a multimédii.

3.1.1.1 Widgety a okna

Widgety představují základní stavební kameny grafického uživatelského rozhraní v rámci Qt. Každá součást GUI (tlačítko,textový editor, atd.) představuje widget, který je umístěný na určitém místě v rámci okna, případě sám o sobě představuje samostatné okno. Konkrétní typ widgetu je definován jako podtřída třídy QWidget, jež je sama o sobě podtřídou třídy QObject (základní třída pro všechny objekty v rámci Qt). QWidget není abstraktní třídou a může být použit jakožto kontejner pro další widgety. To tedy mimo jiné znamená, že konkrétní widget může mít za rodiče jiný widget. To se v praxi projevuje tak, že tento je umístěn do prostoru vyhrazeného rodičem a jeho pozice se tedy vztahuje relativně k rodičovskému widgetu. Pokud je pak rodič odstraněn, jsou odstraněny i všechny widgety, který tento obsahoval.

Widgety, jež nejsou vloženy do nadřazeného rodičovského widgetu, představují samostatná okna. Tyto pak obvykle mají rám a titulní lištu, ačkoliv je možné vytvořit i okno bez těchto částí. Mimo možnost vytvoření okna pomocí QWidgetu bez rodiče existují v Qt dvě základní třídy; QMainWindow a QDialog. QMainWindow představuje, jak již název napovídá, hlavní okno s vlastním layoutem (rozestavení dílčích wdigetů), nástrojovou lištou, stavovým řádkem a zejména pak jednotlivými widgety. QDialog se poté používá jako sekundární okno, které uživateli poskytuje nějakou explicitní informaci (např. stav aplikace, chyba) a případně mu prezentuje určitou volbu mezi konkrétními možnostmi.

3.1.1.2 Signály a sloty

Další důležitou částí frameworku Qt, která je v rámci implementace hojně využívána, je metoda signálů a slotů používaná pro komunikaci mezi jednotlivými objekty v rámci GUI. Signál je vytvořen a vyslán v případě, že se stane nějaká předem definovaná událost (kliknutí na tlačítko, změna rozlišení okna atd.). Qt widgety pak mají typicky několik základních signálů dle specifického typu objektu. Opakem signálu je poté slot, což je tedy funkce, která je volána v reakci na konkrétní signál. I v tomto případě existuje několik předdefinovaných slotů, ke kterým lze však přidávat sloty vlastní. Aby bylo jasně definované, který slot patří ke kterému signálu, musí být tyto propojeny prostřednictvím funkce Connnect. Platí pak, že jeden slot může spouštět více signálů, stejně tak jako jeden signál může zavolat několik slotů. Alternativně, jelikož slot i signál jsou nezávislé komponenty, nemusí být tyto propojeni s ničím, částečně pak ale ztrácejí na významu.

3.1.2 QCustomPlot

QCustomPlot [41] je vykreslovací knihovna pro Qt vyvíjena Emanuelem Eichhammerem, jež se primárně zaměřuje na vytváření kvalitních 2D grafů a diagramů. Díky vysoké výkonnosti je navíc vhodná i pro vizualizaci v reálném čase.

Použití této knihovny v rámci implementace poskytuje hlavní výhodu v jednoduchém přenesení grafové funkcionality prostřednictvím třídy QCustomPlot do individuálních widgetů, které pak podléhají stejné logice, jako jiné objekty v rámci Qt. Navíc, jednotlivé grafy disponují velmi efektivním vytvářením základních prvků, jakými je třeba legenda, případně manipulaci s jednotlivými částmi grafu (měřítka souřadných os, názvy, grafová mřížka, layout v rámci grafu, atd.). Významným přínosem je poté taktéž hotová implementace Zoomu and Pan v rámci knihovny. Stačí tedy pro příslušný graf tyto metody pouze povolit, načež je ještě potřeba pomocí signálů a slotů omezit možnost přesunu mimo původní graf.

Přestože knihovna QCustomPlot disponuje velikým množstvím vizualizač-

ních technik a jejich modifikací, její součástí není přímá implementace složeného plošného grafu. V tomto směru bylo tedy zapotřebí, jak již bylo zmíněno v rámci návrhu, tuto skutečnost brát v potaz a implementovat složený plošný graf pomocí standardního plošného grafu. Další potenciální nevýhodou je poté striktní používání výhradně karteziánské souřadné soustavy pro všechny techniky (nelze tedy prostřednictvím QCustomPlotu implementovat například kruhové diagramy). Poměrně úzké provázání jednotlivých knihovních částí pak rovněž vede k horším možnostem ze strany její úpravy. V neposlední řadě taktéž nutnost spuštění knihovny v hlavním vlákně aplikace může v některých případech vést k nižší potenciální výkonnosti výsledku.

3.1.3 Verze vybraných technologií

Na tomto místě rovněž zmiňujeme pro případ navazujících práci a zachování potenciální kompatibility verze jednotlivých technologií, které byly při implementaci použity:

- Qt 5.7.0
- Qt Creator 4.1.0 (Qt IDE)
- QCustomPlot 2.0.0
- C++ 11

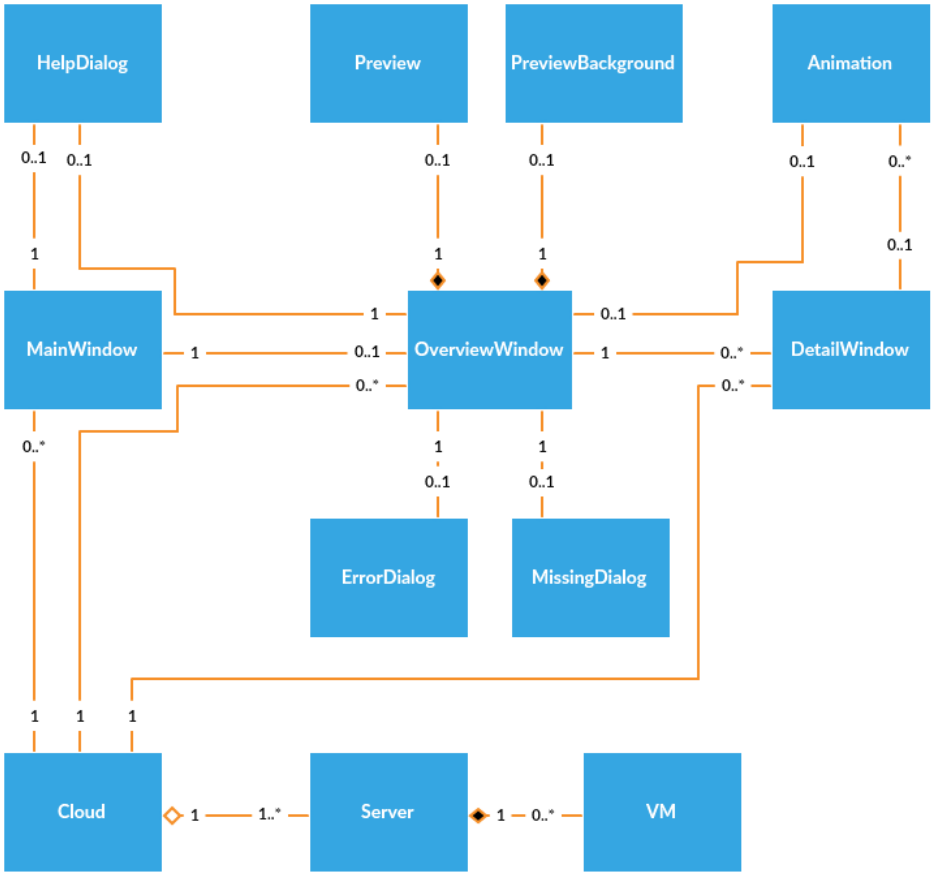
Dále rovněž zmiňujeme základní parametry počítače, na kterém byla aplikace vyvíjena, a to i přesto, že tyto by pravděpodobně neměly nijakým způsobem vlastní implementaci ovlivnit (s možnou výjimkou volby OS):

- OS 64-bit Windows 10
- RAM 16GB
- Procesor Intel Core i5-6500 4x3.20GHz (3.60GHz turbo)

3.2 Rozdělení vizualizační aplikace

Z hlediska implementace můžeme rozdělit vizualizační aplikaci do několika základních tříd, jež obstarávají dílčí funkcionalitu. Předně jsou to tři základní třídy, které kopírují strukturu cloudu a obsahují vizualizovaná data - třída ***Cloud***, ***Server*** a ***VM***. Dále jsou to třídy, jež reprezentují jednotlivé pohledy, tedy vlastní okna v rámci aplikace. Kromě hlavního okna s třídou ***MainWindow***, ve kterém uživatel provádí počáteční selekci časového intervalu a typu zatížení, je to dále přehledové okno a třída ***OverviewWindow*** a detailní

okno s třídou ***DetailWindow***. V přehledovému oknu jsou následně přidruženy dvě třídy ***Preview*** a ***PreviewBackground*** reprezentující náhled a kontext v rámci přehledové heatmapy. Velmi důležitá je rovněž i třída ***Animation***, která vytváří jednotlivé framy pro všechny animace v rámci aplikace. V neposlední řadě jsou zde i tři třídy pro dialogová okna, jež uživatele informují o chybě či obsahují nápovědu, tedy ***ErrorDialog***, ***HelpDialog*** a ***MissingDialog***. Základní struktura je vyobrazena na následujícím diagramu:



Obrázek 3.1: Diagram tříd vizualizační aplikace

Jak jednotlivé třídy dialogů, tak třídy oken pochopitelně dědí z rodičovských Qt tříd, tedy QDialog, potažmo QMainWindow. Náhled pak představuje samostatný graf, tedy Preview a PreviewBackground jsou potomky třídy QCustomPlot.

3.3 Funkcionalita tříd a jejich metod

Na tomto místě stručněji popíšeme funkce jednotlivých tříd a jejich vybraných metod. Pro podrobnější přehled a pochopení jednotlivých částí vizte vlastní zdrojový kód s příslušnými komentáři.

3.3.1 VM

Třída reprezentující jeden virtuální stroj. Hlavní součástí jsou dvě mapy reprezentující procesorové, respektive paměťové zatížení, kde klíč je časový údaj (timestamp) a hodnota je vlastní zatížení pro daný okamžik. Základními metodami jsou především gettery a settery, jež předávají konkrétní údaje do mapy např.:

```
bool AddMemoryTimestamp(double time, double value)
```

V konstruktoru třídy se pak předává odkaz na supervizora (server), jméno a maximální hodnoty pro virtuální paměť a procesor. Kvůli absenci backendové části (viz. níže), jsou dočasně součástí této třídy i dvě metody pro agregaci dat:

```
void AggregateProcessor(double start, double end, int
    numOfValues)
void AggregateMemory(double start, double end, int
    numOfValues)
```

Tyto metody obsahují implementaci algoritmu 10 a jejich funkcionalita tedy pak bude přesunuta právě na backendovou část aplikace k odstínění vznikajícího zatížení při výpočtu. Třída pak dále ještě obsahuje metody

```
double calculateProcUsage(double vm_usage)
double calculateMemoryUsage(double vm_usage)
```

pro případ nutnosti přepočítání zatížení relativně vůči reálnému procesoru/-pamětem supervizora (percentuální zatížení), jež implementují algoritmus 3.

3.3.2 Server

Třída, která reprezentuje konkrétní server, tedy supervizora v rámci cloudu obsahující konkrétní VM. Součástí je tedy vektor konkrétních virtuálních strojů jež serveru náleží a metody

```
bool AddVirtual(VM* virt)
bool RemoveVirtual(QString name)
```

Kromě tohoto má server ještě svoje vlastní mapy pro celkové procesorové a paměťové zatížení, přičemž tyto nasčítává za použití metod

```
void CalculateMemoryUsage()
void CalculateProcUsage()
```

pro sumarizaci zatížení pro všechna data (všechny časové údaje), případně pomocí metod

```
void CalculateMemoryUsage(double from, double to)
void CalculateProcUsage(double from, double to)
```

pro celkové zatížení přes vybraný časový interval. V konstruktoru je opět předáno jméno serveru (ID), odkaz na cloud a maximální hodnoty pro skutečnou paměť a procesor serveru.

3.3.3 Cloud

Třída reprezentující celý serverový cloud. Jako taková má k dispozici přístup k jednotlivým serverům (vektor serverů) a metodám pro jejich přidávání a odebírání:

```
bool AddServer(Server* serv)
bool RemoveServer(QString name)
```

Prostřednictvím jednotlivých serverů má pak pochopitelně přístup i k jednotlivým VM. Uchovává údaje o maximálním a minimálním timestampu, který se v rámci dat vyskytuje (jednoznačné ohraničení maximálního možného časového intervalu). Za tímto účelem má k dispozici metodu

```
void CalculateMinMaxTimestamps()
```

která minimální a maximální timestampy nalezne. Důležitou metodou je poté

```
void CalculateFromTo(double& from, double& to, bool mem)
```

jejíž funkcí je nalézt nejbližší levý (from), respektive pravý (to) timestamp ke zvoleným časovým okamžikům, jež uživatel zadá. Nalezne tedy nejbližší časové okamžiky, pro která existují reálná data ve vztahu k zvolenému typu zatížení.

3.3.4 MainWindow

Třída reprezentující hlavní okno, v rámci kterého uživatel volí typ vizualizovaného zatížení a časový interval. Třída tedy primárně na základě zvolených parametrů vybere tu část dat, která této volbě odpovídá a tu nadále předá k vizualizaci. Rovněž je zde uchováván údaj o tom, zdali bude potenciální detailní pohled dynamický, či statický. V rámci konstruktoru dojde k nastavení celého GUI, tedy jednotlivých položek, potažmo dojde k vypočtení maximálního časového rozsahu pro vybraná data.

Další metody v rámci této třídy pak představují zejména jednotlivé sloty, které reagují na změny v rámci GUI (zmáčknutí tlačítka, změna času v časovém editoru, atd.). Příkladem mohou být metody

```
void on_dateFrom_dateTimeChange(const QDateTime &dateTime)
void on_dateTo_dateTimeChange(const QDateTime &dateTime)
```

které kromě nastavení ohraničujícího intervalu pro vizualizaci pak navíc ošetřují nestandardní případy, kdy je kupříkladu vybraný počáteční časový údaj *from* větší než konečný časový údaj *to* a naopak.

Po stisknutí tlačítka *Show in new window* je poté zavolán slot

```
void on_monitoringButton_clicked()
```

jež vytvoří na základě zvolených údajů nové přehledové okno, tedy objekt třídy OverviewWindow.

3.3.5 OverviewWindow

Třída reprezentující přehledové okno. Hlavní částí je tedy pochopitelně přehledový graf, jež představuje vizualizační techniku heatmapy. Z hlediska grafu jsou základní dvě metody:

```
void initOverview()
```

Tato metoda inicializuje přehledový graf, tedy nastaví jednotlivé osy, jejich pojmenování a měřítko a rozsah. Druhou podstatnou metodou je poté

```
void plotOverview()
```

Tato metoda zpracuje data, a to tím způsobem, že na základě jejich hodnot vytvoří jednotlivé dílky heatmapy, které poté v korespondenci s jednotlivými servery (osa Y) a danými časovými okamžiky (osa X) umístí na správné místo. Následně pak upraví rozsahy os podle rozsahu výsledné heatmapy. Jedná se tedy o implementaci algoritmů 2 a 4. Tuto metodu lze pak opakovaně volat při překreslování grafu.

Velmi důležitou součástí je i implementace slotu

```
bool eventFilter(QObject *target, QEvent *event)
```

jež je volán v případě, že dojde k interakci s vlastním grafem (kliknutí na graf). Obsahem metody je tedy zjištění místa, do kterého v grafu uživatel kliknul (algoritmy 9 a 5) a následná selekce/reselekce/deselekce serveru pro případ náhledu a detailního pohledu. Rovněž zde dochází v závislosti na interakci k vytvoření náhledového grafu a kontextového grafu, viz. třídy Preview a PreviewContext.

Při změně stavu, tedy vybraného serveru, je pak prostřednictvím vytvořeného časovače a slotu

```
void timerSlot()
```

spuštěna příslušná animace, přičemž tento slot je volán v pravidelných intervalech (15ms) a postupně překresluje celý graf na základě přepočítaných dat z třídy Animation, tedy až do koncového framu.

Mezi další významné sloty pak patří kupříkladu

```
void resizeEvent(QResizeEvent* event)
```

který ošetřuje změnu rozlišení okna a tedy i správné přepočítání umístění jednotlivých částí, respektive náhledu a náhledového kontextu, jež jsou brány jako separátní grafy a jejichž umístění je tedy třeba aktualizovat.

V neposlední řadě pak metoda

```
void plotLines(bool from, bool to, double key)
```

vykresluje čáry v přehledovém grafu, jež vyznačují zvolený časový podinterval, který uživatel mění za pomocí šoupátek v horní části obrazovky. Změna jejich pozice a tedy i krajních hodnot podintervalu pak zpracovávána pomocí slotů

```
void on_sliderTo_valueChanged()
void on_sliderFrom_valueChanged()
```

3.3.6 Preview

Třída Preview reprezentuje náhledový graf, jež může být součástí přehledového grafu v rámci OverviewWindow a který pomocí techniky složeného plošného grafu zobrazuje náhled toho, co se v případě zobrazení detailního okna ukáže v jeho hlavním grafu (v případě náhledu je řazení VM dle zatížení implicitně od max do min bez možnosti změny, to je umožněno až v rámci detailu). Třída je tedy realizací zmiňovaného postupu Focus and Context. V rámci konstruktoru je striktně nastaveno rozlišení (výška, šířka) celého grafu, které se potenciálně může měnit v závislosti na úpravě rozlišení přehledové heatmapy. Rovněž jsou třídě předána data, která jsou určena vybraným podintervalem, pro který je náhled vytvořen. V metodě

```
void preparePreviewData()
```

dochází k přípravě dat pro vizualizaci (vytvoření jednotlivých VM pro graf, řazení podle max). Platí, že jak v náhledu, tak později i detailu je vizualizováno maximálně 20 VM, jež jsou vybrány na základě řadícího kritéria (pro více VM je obtížné najít rozlišitelné barvy, navíc tyto s největší pravděpodobností již nejsou v závislosti na kritériu významné). Výjimkou je vyhledávání konkrétního VM, kdy je tento zařazen i když je jeho význam z hlediska zatížení malý. Součástí této metody je tedy i implementace algoritmu 1. Následně je pak prostřednictvím metody

```
void initAreaChart()
```

vykreslen náhledový složený plošný graf, přičemž z grafu jsou odstraněny všechny popisky, legenda i osy tak, aby výsledný graf "zapadl" do přehledové heatmapy a byl s ní vizuálně konzistentní.

Součástí třídy je i metoda

```
void initColours(bool find)
```

která na základě toho, jestli je návrh vytvořen v rámci vyhledávání konkrétního VM či nikoliv, upraví barvy pro jednotlivé plochy (při vyhledávání jsou plochy všech VM mimo hledaného šedivou barvou, vyhledávaný pak červeně).

3.3.7 PreveiwBackground

Tato třída představuje kontext k náhledu, jinými slovy reprezentuje tedy plošný graf, jehož silueta představuje celkové zatížení daného serveru. Kromě barevného rozlišení, které zůstává stejné jako u původního řádku heatmapy, přibude dále ještě rozlišení stran výšky plošného grafu v daném časovém okamžiku. Rozdíl oproti náhledu je tedy ten, že v rámci kontextového grafu se pracuje pouze s celkovým součtem zatížení (proto plošný, nikoliv složený plošný graf). Dalším rozdílem je následně implicitní ohraničení, které je pro kontext vždy dané maximálním rozsahem osy X v přehledovém grafu (u náhledu je to dáno uživatelem zvoleným podinteravalem).

Z hlediska metod tato třída obsahuje, obdobně jako Preview, metodu pro předzpracování dat

```
void preparePreviewBackgroundData()
```

Její obsah se však liší právě v absenci zpracování zatížení pro konkrétní VM (jednoduše stačí dílčí zatížení sečíst pro daný timestamp). K vykreslení grafu je pak použita opět metoda

```
void initAreaChart()
```

která však v tomto případě vytváří plochy ne v závislosti na VM, ale na elementárním časovém úseku (mezi dvěma sousedními timestampy) tak, aby došlo k rozdělení plošného grafu v korespondenci s dílky heatmapy. Tyto jsou pak vybarveny barvami dle zatížení, obdobně jako v případě přehledového grafu. Tato metoda tedy implementuje algoritmus 6.

3.3.8 DetailWindow

Tato třída představuje detailní pohled, tedy okno, jež obsahuje dva hlavní grafy - složený plošný graf a složený sloupcový graf, ve kterém jsou detailněji vizualizována data pro vybrané časové okamžiky. Z hlediska logiky a dílčích funkcionalit je tedy tato třída nejobsáhlejší.

V rámci konstruktoru jsou třídě předána data pro validní časový interval, který byl vybrán u přehledového okna. Rovněž jsou zde pro oba grafy zapnuty možnosti interakce Zoom and Pan, přičemž aby se uživatel nemohl v rámci pohybu po grafu dostat mimo původní rozsah, je tento pohyb prostřednictvím slotů

```
void on_AXrange_changed(QCPRange newRange)
void on_BXrange_changed(QCPRange newRange)
void on_AYrange_changed(QCPRange newRange)
void on_BYrange_changed(QCPRange newRange)
```

omezen. Tyto jednotlivé sloty jsou volány signály pro změnu měřítka jednotlivých os, načež je tedy zkontrolováno, zdali nedošlo k překročení určené maximální hodnoty.

Mezi důležité metody patří dále zejména

```
void initAreaChart()
void initDetail()
```

které, jak již název napovídá, inicializují oba dva grafy do výchozí, prázdné pozice, kdy dojde k nastavení základních parametrů grafů (jména os, rozsahy os, měřítko os, pozice legendy, vzdálenost okrajů atd.). Následně

```
void replotAreaChart()
void replotDetail()
```

slouží k překreslení (i prvotnímu vykreslení) jednotlivých grafů na základě dat, jež jsou v rámci těchto metod v souvislosti se zvoleným grafem vhodně upraveny (vytvoření dílčích sloupců podle dat, řazení VM podle celkového zatížení v závislosti na kritériu, vytvoření ploch pro složený plošný graf - algoritmus 1). Opět zde platí, že vykreslujeme nejvýše 20 VM dle daného kritéria (min/max), zbylé jsou marginální a další barevná kategorizace by byla obtížná (výjimkou je varianta přímého výběru vyhledáváním).

Z hlediska interakce je zde opět zcela zásádní implementace slotu

```
bool eventFilter(QObject *target, QEvent *event)
```

V této metodě se nejprve zjistí se kterým ze dvou grafů uživatel interaguje (na základě globálních souřadnic okna), následně pak jakým tlačítkem myši uživatel kliknul - levé tlačítko signalizuje výběr VM, pravé pak, v případě horního grafu, výběr časového okamžiku pro detailnější pohled (vytvoří složený sloupcový graf). Na základě této selekce je poté prostřednictvím implementace algoritmů 8 a 9 zvolen konkrétní VM, respektive časový okamžik.

Poté, co je jednoznačně určen výběr, dojde k animaci. Za tímto účelem jsou zde metody

```
void prepareBarsForAnimation()
void prepareAreaChartForAnimation()
void setupAnimation(int position)
```

Zatímco vlastní třída animace se zabývá interpolací konkrétních hodnot, první dvě zmíněné metody připravují pro animaci jednotlivé části vizualizační techniky; dochází k vytvoření nové plochy a sloupců pro vybraný VM (ten má v daném okamžiku dvě plochy a dva sloupce - jeden se zmenšuje, druhý zvětšuje), je dočasně vypnuta interakce tak, aby do průběhu animace uživatel nemohl zasahovat. Třetí metoda pak zařizuje vlastní předání dat do animační třídy a vytvoření časovačů, které jsou následně spojeny se sloty

```
void timerSlot()
void timerSlotStackedBar()
```

Tyto, obdobně jako v případě OverviewWindow, překreslují v pravidelných intervalech jednotlivé grafy na základě interpolovaných dat z třídy Animation.

Animace je vytvořena i v případě zavolání slotů

```
void on_minMaxBox_activated(const QString &arg1)
void on_resetButton_clicked()
```

pro změnu řadícího kritéria prostřednictvím combo boxu, respektive uvedení pohledu do výchozího stavu po stisknutí tlačítka *Reset*.

Při selekci jednotlivých časových okamžiků a vytvoření složených sloupců rovněž dochází k vykreslení spojovacích čar mezi těmito sloupci a korespondujícími body na ose X u horního složeného plošného grafu. Tuto funkcionalitu obstarávají metody

```
void prepareConnectingLines()
void drawConnectingLines(QPainter *p)
```

jež v prvním případě nalezne krajní body a v druhém případě tyto spojí čárou. Pro nalezení bodů je potřeba využít metodu

```
void transformPointToGlobal(double key, QPoint &fromPoint,
    QPoint& toPoint, int i)
```

kdy relativní souřadnice v rámci obou grafů jsou převedeny na globální pro celé okno (QPainter, který vykresluje čáry, pracuje s globálními souřadnicemi okna). Jedná se tedy o implementaci algoritmu 5.

Při výběru časového okamžiku pro detail jsou rovněž pod jednotlivými složenými sloupci vytvořena tlačítka, která výběr daného okamžiku zruší, a to prostřednictvím slotu

```
void on_deleteButton_clicked()
```

Přemístění těchto tlačítek do výchozí pozice je poté obstaráváno metodou

```
void moveButtons()
```

Jak pozice spojovacích čar, tak tlačítek pro rušení vybraných timestampů je určena relativně k celému oknu. Je tedy potřeba tyto v případě změny rozlišení aktualizovat, k čemuž opět slouží slot

```
void resizeEvent(QResizeEvent* event)
```

V neposlední řadě třída také obsahuje metody

```
void setDateLabel()
void addDateLabel()
```

pro určení a následné vykreslení části datumu, jež je pro celý zvolený interval stejný. Díky tomu je pak tedy zbytečné tuto část vypisovat k jednotlivým časovým bodům na osách X.

3.3.9 Animation

Třída, jejíž hlavním funkcí je na základě vstupních dat tato připravit a následně interpolovat do vybraného počtu framů. Součástí jsou metody

```
void PrepareDataPriority()
void PrepareDataSwap()
void PrepareDataStackedBar()
void PrepareDataOverview()
```

jejichž funkcí je data, která byla do třídy předána z dílčích částí aplikace (OverviewWindow a DetailWindow), nejprve připravit pro následnou interpolaci. V praxi to znamená, že jsou vytvořena upravená data pro počáteční a koncový frame, přičemž počáteční frame částečně koresponduje s daty předanými (je třeba i tyto částečně upravit), koncový frame je poté dopočítán na základě parametrů předaných třídě (zvolený prioritní VM, selektovaný server, překlopení min a max). Pomocí následujících metod

```
void Interpolate()
void InterpolateStackedBar()
void InterpolateOverview()
```

poté dojde k interpolaci mezi jednotlivými framy (počet framů, výchozí data a typ animace jsou předány v konstruktoru třídy). Tyto metody tedy implementují algoritmus 7 s tím, že interpolace dat se u jednotlivých grafů díky jejich rozdílné podstatě částečně liší (proto separátní metody). Výsledek je poté uložen do vektoru, který obsahuje data pro jednotlivé framy a jež je poté volně přístupný za účelem vykreslení.

3.3.10 Dialogy

Třídy dialogů se v zásadě nijakým způsobem neliší od implicitní třídy QDialog, spíše v jejich rámci dochází k úpravě GUI výsledných informačních dialogových oken, vzhledu textu a vlastnímu zdělení, jež se liší v závislosti na třídě daného dialogu.

3.4 Sběr dat a backend

V rámci implementace propojení aplikace s vlastním cloudem a tedy umožnění sběru reálných dat je v prvé řadě třeba poděkovat Ing. Jiřími Chludilovi, který na základě našeho návrhu vytvořil REST API, jehož prostřednictvím lze přistoupit k jednotlivým serverům a získat požadovaná data o nich a jejich VM. Bohužel, ve zbývajícím časovém horizontu nebylo možné zprovoznit dostatečně funkční backendovou část aplikace, která by získávala z cloudu tato data a dále je zpracovávala. Z tohoto důvodu byla pro celkové testování aplikace použita uměle vygenerovaná data, jež chod cloudu věrně simulují. Část funkcionality backendové části pak byla přesunuta do vlastní vizualizační aplikace. Na závěr je třeba říci, že do budoucna bude dokončení backendu prioritním cílem navazujících prací a pozdější nasazení aplikace na cloud by tedy nemělo představovat větší problém, zejména pak díky vyšší fázi rozpracovanosti této části.

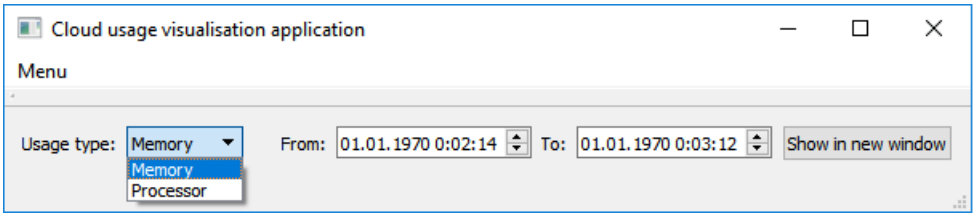
Kapitola 4

Výsledky

V této kapitole si krátce ukážeme výslednou vizualizační aplikaci prostřednictvím screenshotů a popíšeme funkcionality jednotlivých částí, které do velké míry odpovídají vytvořeným návrhům.

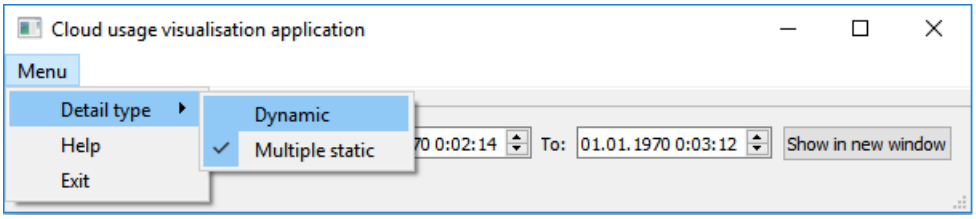
4.1 Hlavní okno

V rámci hlavního okna, které se zobrazí po spuštění aplikace, si uživatel vybírá výchozí časový interval (pro volně zadané časové okamžiky jsou nalezeny nejbližší s reálnými daty) a typ sledovaného zatížení cloudu (procesorové nebo paměťové zatížení). Základní vzhled hlavního okna je vidět na následujícím screenshotu:



Obrázek 4.1: Výchozí okno pro vizualizační aplikaci

Kromě této hlavní funkcionality je zde k dispozici i menu, které poskytuje nápovědu a zejména pak možnost změny typu detailu. Tyto základní typy jsou dva; buďto varianta s jedním dynamickým detailem, který se přizpůsobuje volbě v rámci přehledu, nebo více statických detailů.



Obrázek 4.2: Volba typu detailu v rámci hlavního okna

Poté, co uživatel příslušně navolí základní parametry, si prostřednictvím tlačítka *Show in new window* zobrazí přehledové okno.

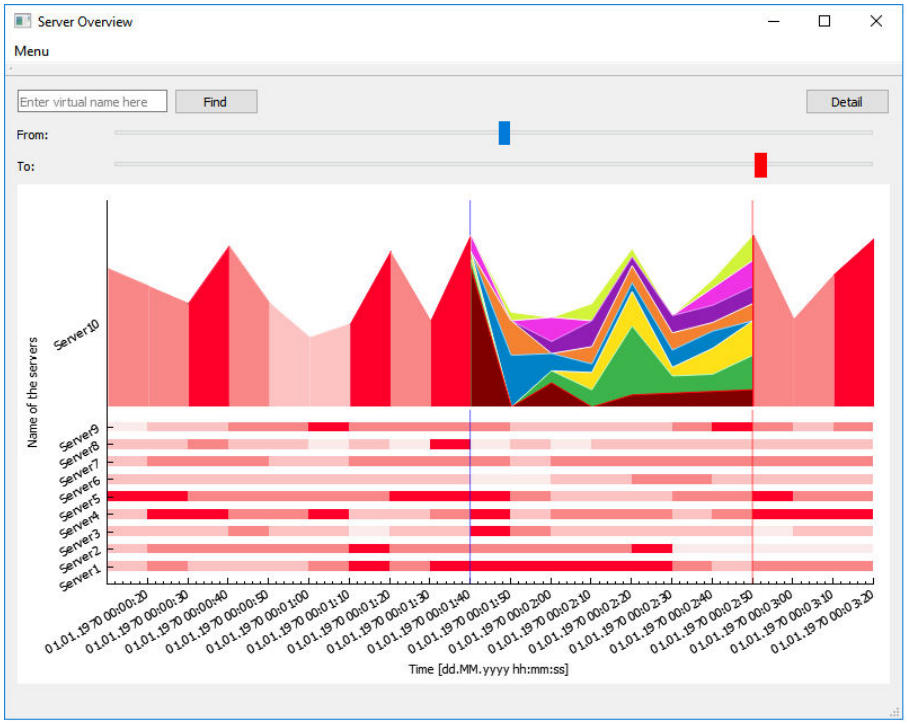
4.2 Přehled

V rámci přehledového okna má uživatel k dispozici pohled na celkové zatížení jednotlivých serverů cloudu skrze techniku heatmapy. V horní části jsou poté k dispozici šoupátka *From* a *To*, jejichž prostřednictvím lze vybrat časový podinterval pro detailnější zkoumání (v závislosti na pozici šoupátek se bere vždy nejbližší dolní/levý timestamp). Počáteční stav přehledového okna je vidět na následujícím screenshotu:

Obrázek 4.3: Výchozí stav přehledového okna v rámci vizualizační aplikace

Uživatel rovněž může prostřednictvím kliknutí do grafu s heatmapou selektovat konkrétní server (vybere se ten nejblíže). Při selekci serveru a časového podintervalu se poté zobrazí náhled obsahující rozdělení zatížení daného serveru mezi jednotlivé VM (virtuály v rámci náhledu jsou vždy seřazeny zdola nahoru od maximálního zatížení po minimální). Mimo selektovaný podinterval je poté pro vybraný server zobrazen kontext náhledu, tak jak lze vidět na následujících screenshotech:

Obrázek 4.4: Kontext náhledu v rámci přehledového okna (není zobrazen vlastní náhled protože časový podinterval je nulový



Obrázek 4.5: Náhled pro Server10 v přehledovém okně společně s kontextem pro daný server

Prostřednictvím kolonky v horním levém rohu a tlačíka *Find* lze explicitně zadat VM, jež se má v rámci náhledu přehledové heatmapy nalézt a zobrazit. Pokud tento v daném časovém intervalu neexistuje, je zobrazen chybový dialog. V opačném případě je nalezený VM v rámci svého serveru zvýrazněn červenou barvou, viz. následující screenshot:

Obrázek 4.6: Explicitní selekce daného VM v rámci přehledového okna

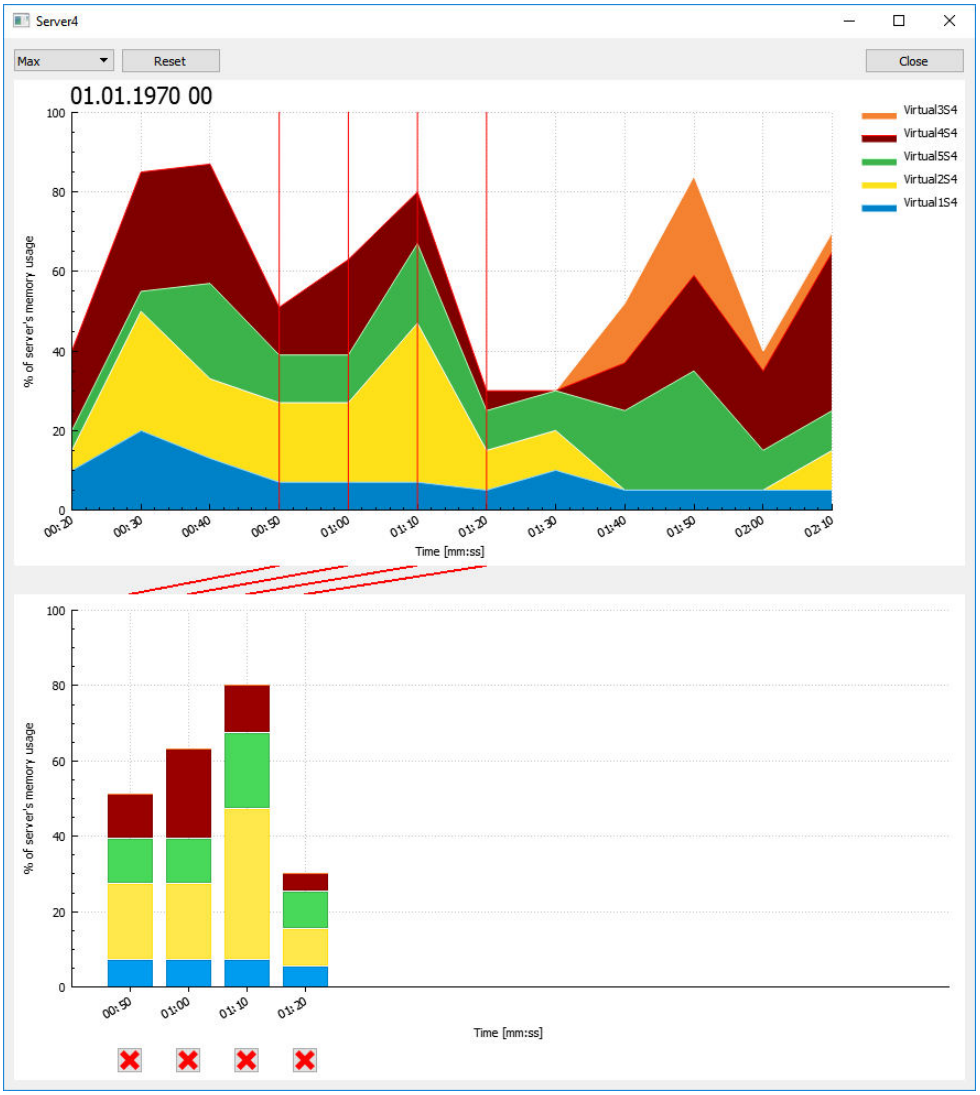
Po selekci serveru a vybrání příslušného časového podintervalu prostřednictvím sliderů lze následně tlačítkem *Detail* v pravém horním rohu vytvořit nové detailní okno.

4.3 Detail

Detailní okno obsahuje v horní polovině složený plošný graf, jež vyobrazuje rozložení zatížení serveru přes zvolený podinterval mezi jednotlivé VM. V levé části hlavičky tohoto grafu je část časového údaje, která je identická pro celý podinterval detailu. Dolní část okna je dále vyhrazena složenému sloupcovému grafu, kde jsou zvlášť vizualizovány vybrané časové okamžiky pro lepší porovnávání poměrů mezi VM. Tyto timestampy jsou vybrány pravým tlačítkem myši v rámci složeného plošného grafu a následně označeny červenou vertikální úsečkou. Složený sloupec pro daný časový okamžik je poté spojen další červenou úsečkou s konkrétním bodem na ose X v rámci horního grafu. Jednotlivé sloupce jsou poté řazeny vždy dle časové osy.

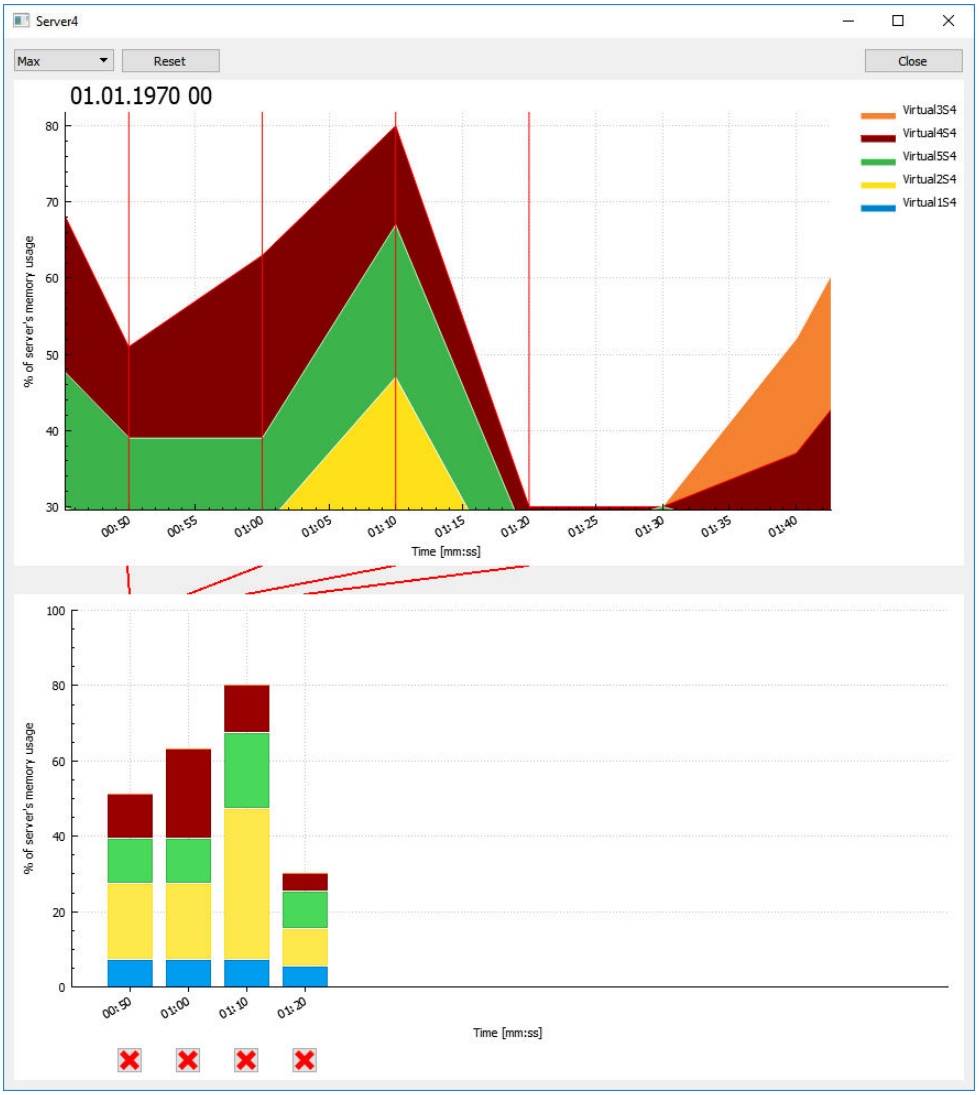
Konkrétní vybraný časový okamžik lze odebrat tlačítkem s červeným křížkem pod daným sloupcem. V horní části okna poté ještě figuruje combo box, jehož prostřednictvím lze měnit řazení VM (max a min), a tlačítko *Reset*, které uvede celé okno do počátečního stavu. Mimo výchozí řazení min a max

lze levým tlačítkem myši v rámci obou grafů selektovat prioritní VM, čímž je tento zarovnán vůči osám X obou grafů (daný VM se posune dolů, ostatní se posunou nahoru). Detailní pohled s několika vybranými časovými okamžiky pak může vypadat následovně:



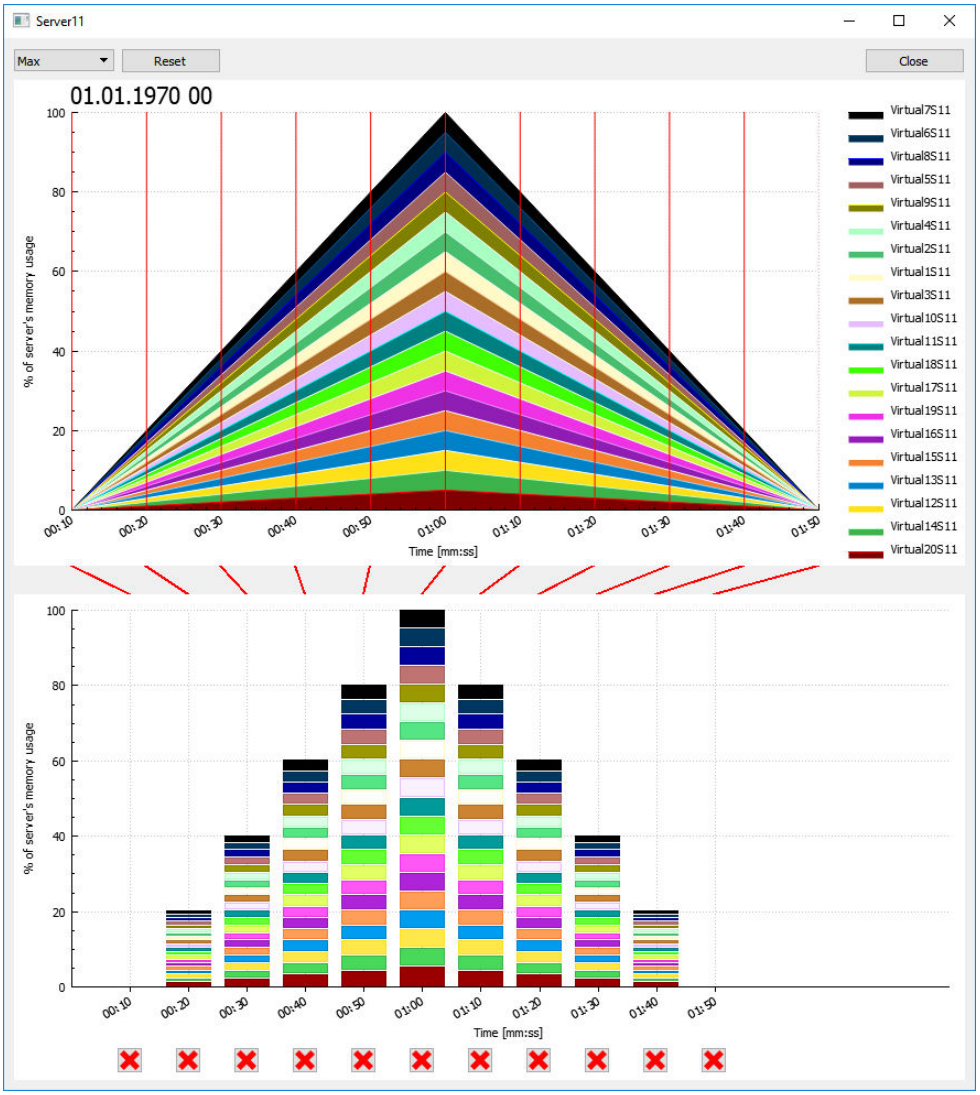
Obrázek 4.7: Detailní pohled s několika vybranými časovými okamžiky

Prostřednictvím kolečka myši následně lze v obou grafech zoomovat a provádět panning. Na dalším screenshotu můžeme vidět přiblíženou předchozí vizualizaci:



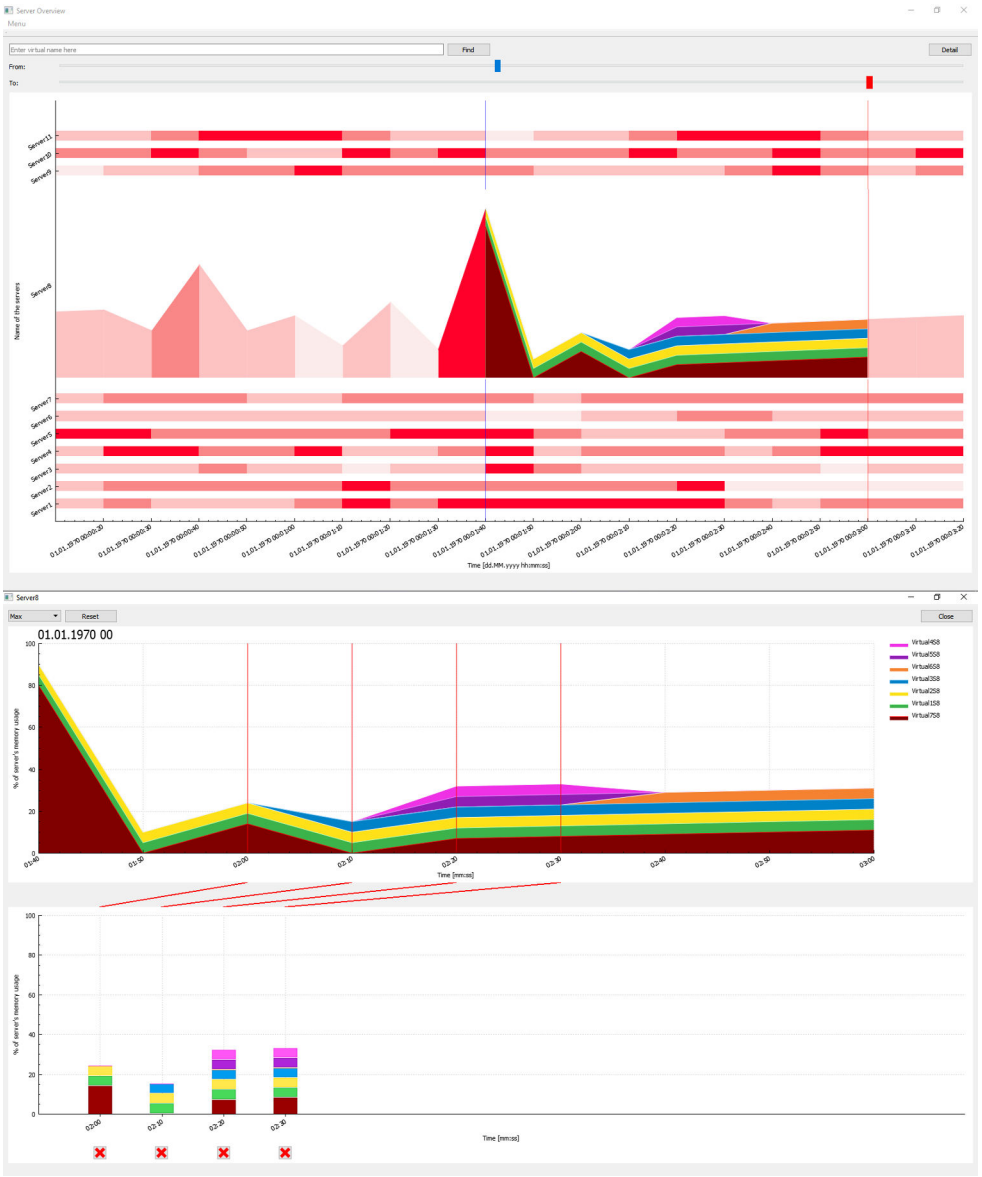
Obrázek 4.8: Přiblížený pohled v složeném plošném grafu

Na následujícím screenshotu můžeme dále vidět maximální možnou škálovatelnost vizualizace ve vztahu k počtu VM, kterých je v takovémto případě dvacet. Předpokládá se, že další VM jsou ve vztahu k danému řazení významově marginální, navíc by již bylo velmi obtížné tyto jednoznačně barevně rozlišit.



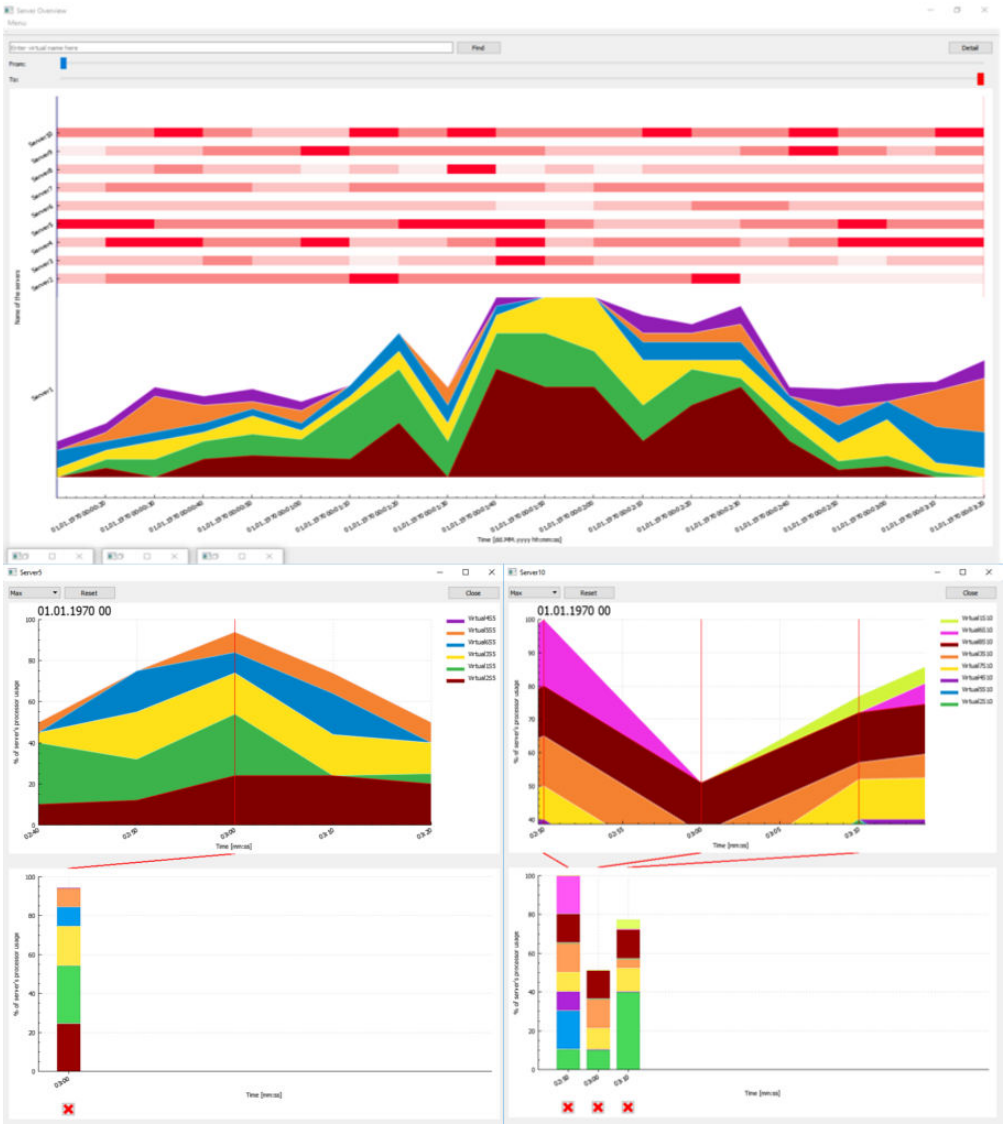
Obrázek 4.9: Maximální možný počet vizualizovaných VM v rámci detailního okna

Na závěrečných dvou screenshotech je poté demonstrován rozdíl mezi typy detailního okna. V prvním případě je detail dynamický; v závislosti na změně výběru serveru či časového podintervalu v přehledu jsou grafy v detailním okně příslušně aktualizovány, přičemž předchozí selekce časových okamžiků je, pokud je to možno (nedošlo ke změně serveru), zachována.



Obrázek 4.10: Přehled s jedním dynamickým detailem

Druhým případem je vícero statických pohledů. Tyto sice nereflektují na změny v přehledu, je však možné jich vytvořit libovolné množství a porovnávat tak naráz detaily pro více serverů či mezi různými časovými podintervaly.



Obrázek 4.11: Přehled s vícero statickými detaily

Kapitola 5

Testování

Předmětem testování bylo ověřit základní použitelnost výsledné vizualizační aplikace, a to jak ze strany výkonu a tedy možnosti zpracování velkého množství dat, tak ze strany uživatelské přívětivosti (použitelnosti) a pokrytí jednotlivých případů užití. Vhledem k tomu, že se bohužel nepodařilo získat reálná data z cloudu (viz. výše), jsou tato simulována uměle vytvořenými. Za účelem testování tedy bylo vytvořeno pět datasetů různé velikosti, přičemž počet serverů pro všechny datasety byl deset a maximální množství VM pro jeden server bylo stanoveno na 40. Každý server pak měl alespoň jeden virtuální stroj. Zvolené datasety měly tedy následující velikosti:

1. dataset - 2 000 záznamů
2. dataset - 6 000 záznamů
3. dataset - 15 000 záznamů
4. dataset - 50 000 záznamů
5. dataset - 200 000 záznamů

Jedním záznamem je poté myšleno jeden časový údaj a k němu korespondující hodnoty paměťového a procesorového zatížení daného VM. Při generování pak byl rovněž v rámci konzistence kladen důraz na to, aby pro všechny časové údaje v datasetu měly všechny VM nějaké zatížení (pokud neměly, bylo jim na závěr nastaveno nulové zatížení).

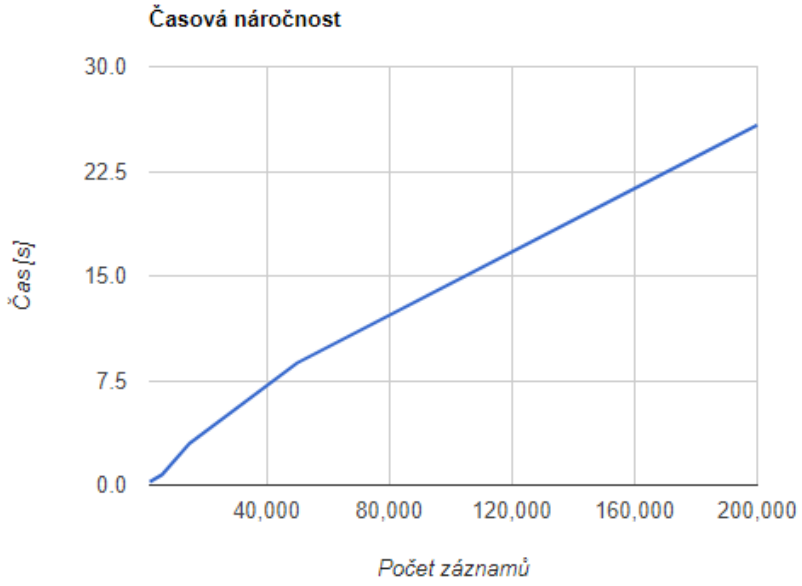
5.1 Testování výkonu

Na základě vytvořených datasetů byla podrobně testována časová a paměťová náročnost aplikace. Za tímto účelem byl použit počítač s následujícími základními parametry:

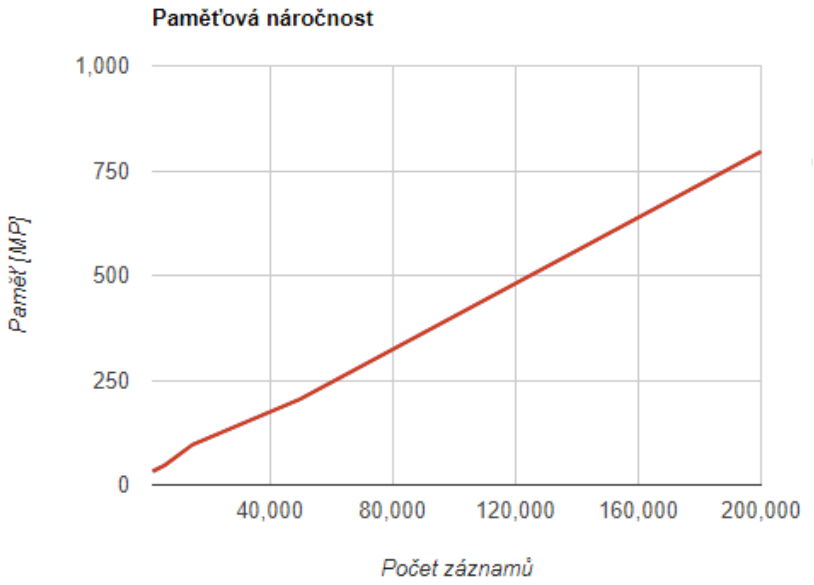
- OS 64-bit Windows 10
- RAM 16GB
- Procesor Intel Core i5-6500 4x3.20GHz (3.60GHz turbo)

Celková paměťová náročnost byla měřena v momentu zobrazeného přehledu přes maximální možný časový interval, selektovaného podintervalu (existence náhledu a kontextu) a jednoho zobrazeného dynamického detailu pro daný podinteraval. Časovou náročnosti byl pak chápán čas, jež je nutný na zpracování daného datasetu (přenos vybraných dat do vizualizační aplikace a následná agregace). Výsledky jednotlivého testování výkonu jsou následující:

Dataset	Časová náročnost [s]	Paměťové vytížení [MB]
1	0.220	32.5
2	0.742	47.2
3	2.995	96.5
4	8.752	205.0
5	25.813	795.0



Obrázek 5.1: Graf porovnání závislosti časové náročnosti aplikace na velikosti datasetů



Obrázek 5.2: Graf porovnání závislosti paměťové náročnosti aplikace na velikosti datasetů

Jak můžeme z jednotlivých grafů vyčíst, jak paměťová, tak časová náročnost poměrně rychle roste s přibývající velikostí datasetů. To by v praxi značně omezovalo použití výsledné aplikace, protože reálná data mohou mít i několik milionů záznamů. Na druhou stranu je však nutné si uvědomit, že pomalé zpracování je způsobeno zejména díky absenci backendové části. Tato by totiž úspěšně odstínila veškeré datové manipulace, které jsou v současné chvíli vykonávany přímo v rámci vizualizační aplikace. Prostřednictvím profilingu byl následně určen jako hlavní důvod zpomalování systému právě proces agregace dat na základě zvoleného intervalu.

Z hlediska zatěžování systému vlastní vizualizace a procesy s ní spojené (interakce, animace) ve všech testovaných datasetech zatěžují systém minimálně, přičemž toto zatížení se s rostoucími datasety mění minimálně. Hlavní příčinou je fakt, že systém vlastní vizualizace pracuje s významně redukovanými daty prostřednictvím agregace (konstantní maximální počet dat) či omezení vizualizace detailu a náhledu na 20 VM. Z těchto důvodu by neměla mít vizualizační aplikace po dokončení backendové části, která se bude plně věnovat manipulaci dat v rámci dedikovaného serveru, s většími datasety výraznější problémy.

5.2 Testování použitelnosti

Testování použitelnosti proběhlo v improvizovaném testovacím prostředí za účasti čtyř participantů. Hlavním cílem bylo proniknout do způsobu uživatelské interakce s aplikací a odhalení potenciálních problémů, na které lze při běžném používání aplikace narazit. Základní personálie participantů byly následující:

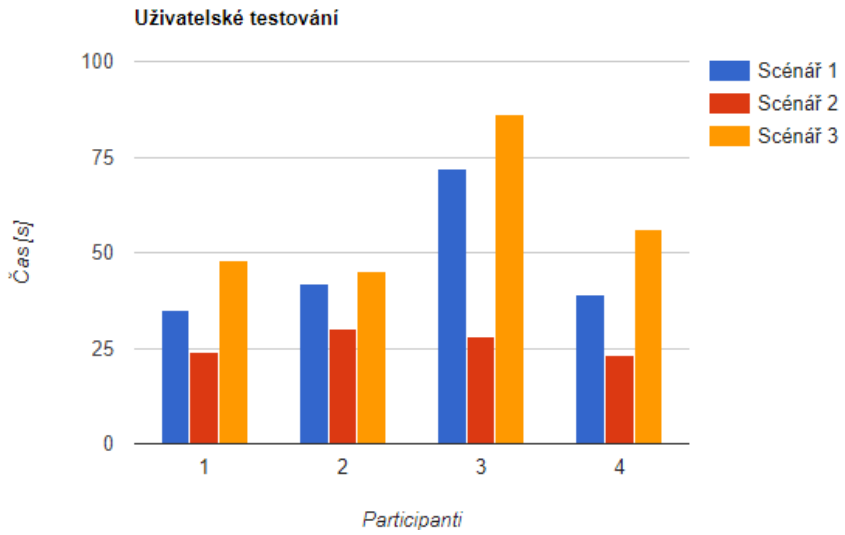
- Participant 1 - Věk: 24, Pohlaví: M
- Participant 2 - Věk: 45, Pohlaví: Ž
- Participant 3 - Věk: 60, Pohlaví: M
- Participant 4 - Věk: 19, Pohlaví: Ž

Každý participant byl nejprve obeznámen se základními vlastnostmi a možnostmi aplikace (GUI tedy nebylo testováno, důraz byl kladen na ověření vlastních vizualizačních technik). V průběhu časového limitu 10 minut pak každý uživatel testoval aplikaci na základě tří testovacích scénářů, jejichž cílem bylo ohodnotit kvalitu pokrytí jednotlivých případu užití. Scénáře byly tedy následující:

- Scénář 1 - Zjistěte, který VM dosáhl v rámci serveru s ID 5456 největšího paměťové zatížení za posledních 24 hodin (pro konkrétní timestamp, nikoliv přes interval).
- Scénář 2 - Zjistěte, v kterých časových intervalech nebyl VM s ID 745223 vůbec procesorově vytížen (nulové zatížení).
- Scénář 3 - Porovnejte zatížení jednotlivých VM pro ten server, který přesáhl v intervalu od 18:00 do 19:00 80% paměťového zatížení. Zjistěte poměry mezi třemi nejméně zatíženými VM pro všechny časové okamžiky v rámci tohoto intervalu.

Pro scénáře 1 a 2 byl použit dataset 1, pro scénář 3 pak dataset 2. Výsledky uživatelského testování (čas v sekundách pro dokonční jednotlivých scénářů) pak byly následující:

Scénář	Participant 1	Participant 2	Participant 3	Participant 4
1	35	42	72	39
2	24	30	28	23
3	48	45	86	56



Obrázek 5.3: Graf časové náročnosti daných scénářů ve vztahu k participantům

Zde je třeba zmínit, že naměřené časy jsou uváděny pouze jako doplněk vlastního testu, přičemž budoucím uživatelům mohou pochopitelně jednotlivé úlohy trvat různě dlouho (čas zde není předmětem podrobnějšího statistického vyhodnocování). Výsledky testování ve skrze potvrdily výhodnost a dobrou použitelnost zvolených vizualizačních technik pro dané případy užití. Participanti dokázali zvolené scénáře úspěšně dokončit ve všech případech, navíc ve většině případů i v relativně krátkém čase. Výjimkou byl scénář 1 a 3 u participanta 3, kde však veskrze došlo spíše k nepochopení daného zadání.

V rámci vlastního testování bylo rovněž objeveno několik nedostatků, které je vhodné do budoucna vyřešit. Předně v průběhu testování hledali někteří participanti možnost vrácení o krok zpět (Undo), která v současné chvíli není k dispozici. Druhým větším nedostatkem pak byla pomalejší volba výchozího časového intervalu v rámci hlavního okna. Tato nedokonalost by mohla být v budoucnu odstraněna prostřednictvím použití kalendářových widgetů pro snadnější selekci konkrétního data.

Pro porovnání byl rovněž testován scénář 2 (s jiným VM) pouze prostřednictvím tabulkového záznamu jednotlivých datasetů, tedy bez jakékoliv vizualizace. V tomto případě byl poté čas úspěšného dokončení v průměru o 700% pomalejší než v případě našeho řešení, což dokazuje nezbytnost nějaké formy vizualizace pro rychlé řešení dané problematiky.

Na závěr je nutné podotknout, že uživatelské testování použitelnosti neproběhlo v dedikovaném prostředí. Výsledná data tedy mohou být tímto faktem negativně ovlivněna. Za účelem podrobnějšího testování je tedy vhodné do budoucna aplikaci podrobit dalšímu uživatelskému testování na širším okruhu participantů a scénářů a ve specializovaném prostředí, například v laboratoři použitelnosti (Usability Lab) na FIT či FEL ČVUT.

Závěr

O práci

Tato práce se ve svém obsahu detailně věnovala problematice vizualizace statických a v čase proměnných hierarchických skalárních dat. Konkrétním příkladem těchto dat, na nichž byla práce primárně stavěna, pak byly údaje o paměťovém a procesorovém zatížení serverového cloudu, respektive jeho dílčích částí.
V práci provedenou analýzu můžeme shrnout do následujících skupin:

- Důležité aspekty obecné vizualizace a práce s ní
- Analýza vizualizačních technik a jejich použitelnosti ve vztahu k zvolené problematice
- Analýza existujících řešení a jejich výhod a nevýhod
- Nutná úprava řešení pro velký objem dat

Tyto jednotlivé skupiny jsou detailně rozebrány v kapitole Analýza. Na základě výsledků z jednotlivých částí byl vytvořen návrh po aplikaci, jejíž cílem je prostřednictvím vhodně zvolených vizualizačních technik a postupů umožnil uživateli jednoduchým způsobem monitorovat vývoj procesorového a paměťového zatížení cloudu a utvářet si vlastní obraz o vývoji trendů prostřednictvím uživatelské interakce. Na základě tohoto návrhu pak byla aplikace s vybranými vizualizačními technikami naimplementována tak, aby došlo k maximálnímu vhodnému pokrytí jednotlivých případu užití. V závěrečné kapitole došlo poté k testování výsledného systému, a to jak ze strany výkonu daného řešení, tak ze strany uživatelské přívětivosti a použitelnosti zvolených vizualizačních postupů.

Úroveň splnění zadání práce

Závěrečné ohlédnutí na zadání diplomové práce a následná diskuze nad způsobem a mírou splnění jednotlivých částí.

- **Analyzujte techniky pro vizualizaci statických a v čase proměnných hierarchických skalárních dat**
  Tomuto úkolu se velmi detailně věnuje první polovina kapitoly Analýza 1, která se individuálně zaměřuje na jednotlivé techniky a vizualizační postupy. Jejich kvalitu pak porovnává na základě schopnosti vhodně přenést informaci ze zvoleného typu dat do vizualizace, zejména s ohledem na zvolenou problematiku časově proměnných hierarchických dat.

- **Analyzujte možnosti aplikace zvolených technik k vizualizaci procesorového a paměťového zatížení cloudu**
  Této části analýzy se věnuje druhá polovina kapitoly Analýza 1, jež nejprve porovnává použitelnost vybraných vizualizační techniky pro problematiku vizualizace zatížení serverového cloudu. Výsledky pak následně konfrontuje se zkoumáním existujících řešení, u nichž jsou sledovány zejména potenciální výhody a nevýhody dílčích aspektů jejich vizualizace. Kombinace výstupů z těchto dvou částí pak jednoznačně určuje výsledné zvolené vizualizační techniky pro návrh a implementaci vizualizační aplikace.

- **Navrhněte aplikaci umožňující monitoring a analýzu procesorového a paměťového zatížení cloudu v určitém časovém intervalu**
  Veškeré vytvořené návrhy výsledné vizualizační aplikace v kapitole Návrh 2 jsou zohledněny vůči výsledkům analýzy. Ve vztahu k vybraným technikám a obecným postupům je zde následně upřesněno množství základních algoritmů, které musí být zohledněny ve výsledné aplikaci tak, aby tato úspěšně řešila zvolenou výchozí problematiku včetně daných případů užití. V této souvislost je předmětem návrhu i funkcionalita dílčích částí aplikace a vzhled jednotlivých pohledů. Na závěr je popsán vytvořený prototyp jež ověřoval použitelnost jednotlivých částí návrhu.

- **Na základě návrhu a vytvořeného prototypu implementujte vybrané postupy do finální aplikace**
  Vlastní implementace vizualizační aplikace je detailněji popsána v kapitole Implementace 3. Výsledná aplikace se zaměřila na zvolené vizualizační techniky heatmapy, složeného plošného a složeného sloupcového grafu. Rovněž byl kladen důraz na kvalitu uživatelské interakce, aplikování základní principů průzkumu dat a celkové zpřehlednění vizualizace. Aplikace z tohoto hlediska pokrývá všechny tři deklarované případy užití

a představuje tak funkční řešení výchozí problematiky. V současné chvíli ještě není zcela dokončena část týkající se přenosu a zpracovávání dat z reálného cloudu, aplikace tedy používá data uměle generovaná. Dokončení této částí je pak prioritním předmětem navazujících prací na tomto projektu.

- **Aplikaci otestujte na pěti datových sadách rozdílné složitosti získaných z reálného cloudu (nebo z jeho simulace) sestávajícího alespoň z deseti serverů a třiceti virtuálních serverů**
  Sekce Testování 5 se podrobněji zabývala kvalitou výsledné aplikace v improvizovaném testovacím prostředí. Z důvodu absence reálných dat bylo testování prováděno na uměle generovaných datech, která však provoz skutečného cloudu věrně simulují. V rámci detailnějšího zkoumání kvality aplikace je vhodné provést další testování v specializovaném testovacím prostředí.

Možná vylepšení

Z hlediska potenciálních vylepšení výsledného systému v rámci navazujících prací je v prvé řadě prioritou dokončení backendové části a tedy umožnění vizualizace reálných dat z cloudu.

Co se týká samotné vizualizační části, je do budoucna vhodné přidat možnost Undo, tedy vrácení se o krok zpět (zrušení posledního kroku). Dalším pozitivní úpravou může být doplnění selekce výchozího časové intervalu o kalendář, prostřednictvím kterého bude jednodušší vybrat jednotlivé rozsahy v rámci dní. Snahou bylo tyto kalendáře zahrnout už do současné aplikace, bohužel widgety jimiž jsou reprezentovány kalendáře v rámci Qt nejsou zcela ideální (selektované dny jsou špatně čitelné), bude tedy třeba tyto částečně přeprogramovat.

Dalším potenciálním vylepšením je nahradit lineární interpolaci v rámci složeného plošného grafu interpolací kvadratickou, čímž bude vzniklý graf vizuálně přívětivější, přičemž jednotlivé zlomy u daných časových okamžiků tak budou nahrazeny plynulejšími přechody pomocí křivek. V současné chvíli je také stále předmětem diskuze, zdali by potenciálně nebylo výhodou zobrazení jak paměťového, tak procesorového zatížení naráz, potažmo jako součástí jednoho grafu. V rámci analýzy se ukázalo, že tato potřeba zde není, avšak do budoucna se může uvažovat i o této variantě jakožto možném způsobu pro rychlejší monitoring.

V neposlední řadě by v budoucnu bylo vhodné vizualizační aplikaci doplnit o rychlý indikátor aktuálního zatížení jednotlivých serverů, například prostřednictvím speciálních barevných indikátorů (kruhů) či tachometrů.

Budoucnost projektu

Po dokončení backendové části a nasazení aplikace na dedikovaný server se pro projekt jako takový naskýtá velká řada možností, kterými ho lze do budoucna rozšiřovat. Předně lze vizualizaci obohatit o další techniky v závislosti na nových případech užití, které mohou být později přidány. Dále by bylo vhodné systém úžeji propojit s aplikací Pavla Podaného, která se věnuje vizualizaci datových toků v rámci stejného serverového cloudu. Za tímto účelem bude potřeba aplikaci rozšířit na další platformy, předně pak vytvořit alternativu pro web a mobilní zařízení. Výzvou do budoucna je pak aplikaci zobecnit na širší škálu cloudových systému a umožnit vizualizování dalších možných metrik a údajů skrze specializované techniky a prohloubení jednotlivých možností interakce.

Zdroje

[1] FUNKHOUSER, H. G. *A note on a tenth century graph.*Osiris, 1, 260–262.1936

[2] TUFTE, E. R.*The Visual Display of Quantitative Information.*Cheshire, CT: Graphics Press.1983

[3] SNOW, John.*On the Mode of Communication of Cholera.*Health Division, United States Agency for International Development.1955

[4] CUENCA, E. & SALLABERRY, A. & WANG, F. Y. & PONCELET, P.*Visualizing Hierarchical Time Series with a Focus+Context Approach.*Information Visualization Conference (InfoVis 2017).2017

[5] CHEN, Chaomei.*Information Visualization - Beyond the Horizon.*Springer.2004

[6] KHAN, M. & KHAN S.*Data and Information Visualization Methods, and Interactive Mechanisms: A Survey.*International Journal of Computer Applications, 1-14.2011

[7] AIGNER, W. & MIKSCH, S. & MULLER, W. & SCHUMANN, H. & TOMINSKI, CH.*Visualizing Time-Oriented Data – A Systematic View.*University of Rostock,University of Education Weingarten,Danube University Krems.2007

[8] TORY, M. & MOLLER, T.*Human factors in visualization research.*IEEE Trans. Visualization and Computer Graphics (TVCG), vol. 10, no. 1, pp. 72-84.2004

[9] ZHANG, Jin.*Visualization for Information Retrieval.*School of Information Studies, University of Wisconsin Milwaukee.2008

[10] MUNZNER, Tamara.*Visualisation analysis and design.*CRC Press.2014

[11] KAINZ, Christian.*Evaluation of Interactive Visualization Methods to Compare Multivariate Heterogeneous Time Series.*Fakultät für Informatik der Technischen Universität Wien.2009.[diplomová práce]

[12] ABDULLAH, Johari.*Investigating interactive visualisation in a Cloud computing environment.*Lincoln University.2010.[bakalářská práce]

[13] AIGNER, Wolfgang.*Interactive Visualization and Data Analysis: Visual Analytics With a Focus on Time* .Fakultät für Informatik der Technischen Universität Wien.2013.[habilitační práce]

[14] WARE, Colin.*Information Visualization: Perception for Design.*Morgan Kaufmann Publishers Inc., San Francisco, CA.2004

[15] KEIM, D. & MANSMANN, F. & SCHNEIDEWIND, J. & ZIEGLER, H.*Challenges in visual data analysis.*Information Visualization Conference (InfoVis 2006).2006

[16] DIX, Alan.*Introduction to Information Visualisation.*University of Birmingham, School of Computer Science.2012

[17] VALENTE, Ronald.*Analysis of virtual environments through a web based visualization tool.*Rochester Institute of Technology.2009.[online].[přístup 15. ledna 2018]. Dostupné z:
`http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1534\&context=theses`

[18] http://mathworld.wolfram.com.*Chernoff Face.*[online].[přístup 11. března 2018]. Dostupné z:
`http://mathworld.wolfram.com/ChernoffFace.html`

[19] https://peltiertech.com.*Chart Busters: Pie Charts Can't Show Trendlines.*[online].[přístup 9. března 2018]. Dostupné z:
`https://peltiertech.com/chart-busters-pie-charts-cant-show-trendlines`

[20] https://datavizcatalogue.com.*Showing Data Over Time.*[online].[přístup 15. ledna 2018]. Dostupné z:
`https://datavizcatalogue.com/search/time.html`

[21] https://www.anychart.com.*Nightingale Rose Chart.*[online].[přístup 10. března 2018]. Dostupné z:
`https://www.anychart.com/chartopedia/chart-type/nightingale-rose-chart`

[22] https://docs.oracle.com/.*Oracle Fusion Middleware Data Visualization Tools Tag Reference for Oracle ADF Faces.*[online].[přístup 10. března 2018]. Dostupné z:

https://docs.oracle.com/cd/E57014_01/adf/tag-reference-dvt/tagdoc/dvt_bubbleChart.html

[23] https://docs.dhtmlx.com/.*Bar Chart.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://docs.dhtmlx.com/chart__dhxbar.html

[24] https://vizzlo.com.*Stacked Bar Chart.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://vizzlo.com/vizzards/recent

[25] https://www.telerik.com/.*TreeMap.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://www.telerik.com/kendo-ui/treemap

[26] https://labwrite.ncsu.edu.*Scatter Plot.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://labwrite.ncsu.edu/res/gh/gh-linegraph.html

[27] http://pandas.pydata.org.*Parallel Coordinates.*[online].[přístup 15. ledna 2018]. Dostupné z:
http://pandas.pydata.org/pandas-docs/version/0.9.1/visualization.html

[28] www.wikipedia.org.*Condegram Spiral Plot.*[online].[přístup 10. března 2018]. Dostupné z:
https://en.wikipedia.org/wiki/File:Condegram_Spiral_Plot.png

[29] www.wikipedia.org.*Javascript.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://cs.wikipedia.org/wiki/Javascript

[30] www.wikipedia.org.*Java.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://cs.wikipedia.org/wiki/Java

[31] www.wikipedia.org.*C++.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://cs.wikipedia.org/wiki/C++

[32] www.wikipedia.org.*Python.*[online].[přístup 15. ledna 2018]. Dostupné z:
https://cs.wikipedia.org/wiki/Python

[33] paldhous.github.io.*Data visualization: basic principles.*[online].[přístup 15. ledna 2018]. Dostupné z:
http://paldhous.github.io/ucb/2016/dataviz/week2.html

[34] http://vmwaretips.com.*Product Review : Vizioncore vFoglight.*[online].[přístup 15. ledna 2018]. Dostupné z:
http://vmwaretips.com/wp/2008/10/30/product-review-vizioncore-vfoglight

[35] *Lpar2rrd.*[software].[přístup 15. ledna 2018]. Dostupné z: `http://www.lpar2rrd.com/VMware-performance-monitoring.html`

[36] *OpsDataStore.*[software].[přístup 15. ledna 2018]. Dostupné z: `https://opsdatastore.com/`

[37] *Veeam.*[software].[přístup 15. ledna 2018]. Dostupné z: `https://www.veeam.com`

[38] *Quest.*[software].[přístup 15. ledna 2018]. Dostupné z: `https://www.quest.com`

[39] *ManageEngine.*[software].[přístup 15. ledna 2018]. Dostupné z: `https://www.manageengine.com`

[40] *Qt.*[software].[přístup 23. května 2018]. Dostupné z: `https://www.qt.io/`

[41] *QCustomPlot.*[software].[přístup 23. května 2018]. Dostupné z: `http://www.qcustomplot.com/`

[42] CARD, S. K. & MACKINLAY, J. D. & SCHNEIDERMAN, B.*Readings in information visualization: using vision to think.*Morgan Kaufmann Publishers Inc., San Francisco.1999

Příloha A

Seznam použitých zkratek

VM Virtual Machine

GUI Graphical User Interface

ZUI Zooming User Interface

OS Operational System

CPU Central Processing Unit

REST Representational State Transfer

API Application Programming Interface

ACID Atomic, Consistent, Isolated, Durable

Příloha B

Slovník cizích pojmů

- **Cloud**
  Model vývoje a používání počítačových technologií, poskytování služeb servery dostupnými z internetu
- **Software**
  Sada počítačových programů používaných v počítači, které provádějí nějakou činnost
- **Timestamp**
  Časová značka, sekvence znaků označující konkrétní časový okamžik
- **Desktop**
  Klasický stolní počítač
- **Frame**
  Obecné označení pro jeden statický obraz (snímek), například z videa
- **Zoom**
  Přiblížení (zoom in), případně oddálení (zoom out) v rámci scény či grafu
- **Pan**
  Posunutí, například v rámci grafu
- **Focus**
  Pozornost, zaměření se na konkrétní část
- **Layout**
  Rozvržení jednotlivých prvků v rámci tiskové či elektronické stránky, případně okna
- **Wireframe**
  Návrh definující obsah a rozmístění funkčních prvků dané aplikace

- **Slider**
  Šoupátko, ovládací prvek grafického uživatelského rozhraní
- **Combo box**
  Výběrové pole, ovládací prvek grafického uživatelského rozhraní
- **Radio button**
  Přepínač, ovládací prvek grafického uživatelského rozhraní
- **Use case**
  Případ užití - seznam kroků, který obvykle definuje interakci mezi člověkem a systémem
- **Preview**
  Předběžný náhled
- **Overview**
  Přehled, přehledové okno
- **Backend**
  Část aplikace, která slouží k administraci a ke zpracování dat
- **Screenshot**
  Snímek obrazovky