

UNIVERZITA HRADEC KRÁLOVÉ
FAKULTA INFORMATIKY A MANAGEMENTU
KATEDRA INFORMATIKY A KVANTITATIVNÍCH METOD



BAKALÁŘSKÁ PRÁCE

Vert.x jako platforma pro webové aplikace

Autor: Michael Kutý

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové, 2014

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a uvedl jsem všechny použité prameny a literaturu.

V Kroměříži dne 5. srpna 2014

Michael Kutý

Poděkování

Rád bych zde poděkoval doc. Ing. Filipu Malému, Ph.D. za odborné vedení práce, podnětné rady a čas, který mi věnoval.

Anotace

Cílem bakalářské práce je představit principy a terminologii platformy na teoretické úrovni a následně je ověřit na praktickém příkladu. Teoretická část se zaměřuje na popis architektury a její hlavní výhody pro distribuované webové aplikace. V praktické části bude vytvořena aplikace pro správu myšlenkových map. Aplikace se nasadí do dvou referenčních instalací. První do prostředí VirtualBox a druhá v prostředí laboratoře CEPSOS při UHK. Text shrnuje základní možnosti frameworku Vert.x pro začínající uživatele, kteří by rádi použili tento framework pro vývoj webových aplikací.

Annotation

EN verze

Obsah

1 Úvod	1
1.1 Cíl a metodika práce	1
1.2 Postup a předpoklady práce	2
2 Platforma Vert.x	3
2.1 Historie	3
2.2 Architektura	4
2.2.1 Jádro	5
2.2.2 Asynchronní model	5
2.2.3 Terminologie	9
2.2.4 Event Bus	11
2.3 API	14
2.3.1 Základní API	14
2.3.2 Kontainer API	15
2.3.3 Polyglot	15
2.4 Clustering	16
2.4.1 Vysoká dostupnost	17
2.5 Porovnání s Node.js	19
2.5.1 Výkon	19
2.5.2 Vlastnosti	21
2.5.3 Závěr srovnání	21
3 Praktická část	23
3.1 Návrh	23
3.1.1 Cíle aplikace	23
3.1.2 Architektura	23
3.2 Vlastní implementace	23
3.2.1 Řídící verticle	25
3.2.2 Editor	26
3.2.3 Integrace s databází MongoDB	27
3.3 Komunikace v reálném čase	29
3.3.1 Akce	31
3.3.2 Události	31
3.3.3 Klientská aplikace	31
3.4 Polyglot vývoj a moduly	32
3.5 Základní software	34

3.5.1	Java	36
3.5.2	Vert.x	36
3.5.3	Databázový server	36
3.5.4	Nasazení produkční služby	36
3.5.5	Interakce s Vert.x	37
3.6	Škálování	38
3.6.1	Vertikální	38
3.6.2	Horizontální	39
3.6.3	Ladění výkonnosti	39
3.7	Vysoká dostupnost	39
3.8	Integrace do stávající Java aplikace	40
4	Závěr	42
4.1	Možnosti dalšího výzkumu	43
4.1.1	Distribuované výpočty	43
4.1.2	Srovnání	43
	Literatura	46
	Přílohy	I

1 Úvod

V současné době existuje nespočet frameworků¹⁾ pro vývoj webových aplikací ve spoustě programovacích jazycích. Výběr takového nástroje pak může být pro danou aplikaci klíčový. Vzhledem k faktu, že je s aplikací po celý životní cyklus, může se s časem stát svazujícím a nedostačujícím. Na reimplementaci však již není čas nebo peníze. Většina webových aplikací tak dříve nebo později narazí na problematiku škálování, kdy je třeba rozložit aplikaci na více serverů ať už pro zajištění vysoké dostupnosti nebo kvůli velké výpočetní náročnosti. Dnes také není nic neobvyklého, že aplikaci najednou začnou navštěvovat tisíce klientů za minutu. Z rychlé a stabilní aplikace se tak může stát často padající aplikace s nepřiměřenou odezvou.

Právě proto, jsem se rozhodl k hlubšímu zkoumání v dané oblasti webových aplikací. V první části bakalářské práce je popsána architektura a jednotlivé technologie, které mě motivovaly k hlubšímu studiu platformy Vert.x. V hlavní části práce následuje návrh, implementace a nasazení jednostránkové aplikace. V závěru je pak shrnutí kladů a záporů platformy.

1.1 Cíl a metodika práce

Hlavním cílem práce bude zjištění zda-li se platforma Vert.x hodí pro vývoj distribuovaných webových aplikací. Vytvoření jednoduchého webového editoru pro správu myšlenkových map. Na této jednoduché aplikaci bude demonstrována architektura a nasazení aplikace na více serverů pro zajištění vysoké dostupnosti. Zdrojové kódy včetně návodu na spuštění aplikace jsou umístěny veřejně na serveru Github²⁾ a na přiloženém médiu.

Je nutné uchopit problematiku platformy Vert.x v širších souvislostech, proto se práce snaží neopomenout všechny technologie, které s Vert.x souvisí, z kterých Vert.x vychází nebo které přímo integruje. V teoretické části bude čtenář seznámen s důležitými filozofiemi, které platforma nabízí.

Cílem teoretické části je tedy popsat jednotlivé architektonické prvky a komponenty platformy, jejich účel či problém, který řeší. V závěru teoretické části bude platforma

¹⁾jeho cílem je převzetí typických problémů dané oblasti, čímž se usnadní vývoj tak, aby se návrháři a vývojáři mohli soustředit pouze na své zadání

²⁾www.github.com/michaelkutty

srovnána s nástrojem Node.js. Srovnání bude obsahovat test výkonnosti a porovnání vlastností.

V praktické části bude vytvořen editor pro správu a tvorbu myšlenkových map. Tyto mapy bude moci upravovat více uživatelů najednou v reálném čase. Budou popsány a vysvětleny aspekty komunikace v reálném čase včetně samotného nasazení webové aplikace na jednotlivé servery, kde bude prověřena funkčnost distribuovaného provozu aplikace v režimu vysoké dostupnosti.

1.2 Postup a předpoklady práce

Práce předpokládá základní znalost programovacího jazyku Java a JavaScript. Teoretická část se neomezuje pouze na nezbytný popis technologií potřebných k realizaci malé jednostránkové webové aplikace. Představuje stručný pohled na celou platformu Vert.x. Teoretická část může být použita jako odraz k hlubšímu studiu daných technologií a konceptů. Praktická část bude prokládána ukázkami kódu nebo příkazy souvisejícími s vývojem webových aplikací. Práce předpokládá znalost základní terminologie související s programováním obecně. Méně zažité pojmy budou vysvětleny poznámkou pod čarou.

Při vývoji webové aplikace budou použity následující softwarové technologie:

- Vert.x 2.1.2: platforma pro vývoj webových aplikací
- MongoDB: dokumentově orientovaná NoSQL³ databáze
- D3.js: framework pro práci s grafy
- JQuery framework pro práci s GUI(Graphical user interface)

³ databázový koncept, ve kterém datové úložiště i zpracování dat používají jiné prostředky než tabulková schémata tradiční relační databáze

2 Platforma Vert.x

Dnešním trendem internetu jsou aplikace, kde klienti komunikují v reálném čase, proto se také často označují jako kolaborativní aplikace. Tyto aplikace s sebou přinesly drastickou změnu potřeb programátorů, na jednotlivé nástroje. Často programátor zjistí, zda-li se mu výběr podařil, v pokročilé fázi vývoje, kdy už nevede cesta zpět. Objevilo se několik nástrojů, které přinesli revoluci na poli webových aplikací mezi než patří například Node.js, Akka nebo Ruby Event Machine. Tyto nástroje si vydobýly své místo nejen pro možnost škálování na desítky tisíc klientů, což není v dnešní době nic neobvyklého.

Vert.x je projekt vycházející z Node.js, který podnítil závod o pokoření C10K¹ problému. Obě platformy poskytují asynchronní API, které si je co do zaměření velice podobné. Node.js, jak již název napovídá je napsán v jazyce JavaScript, zatímco Vert.x je implementován v Javě. Vert.x ale není pouhá reimplementace Node.js do jazyka Java. Platforma má svou vlastní unikátní filozofii a terminologii, která je diametrálně odlišná od Node.js.

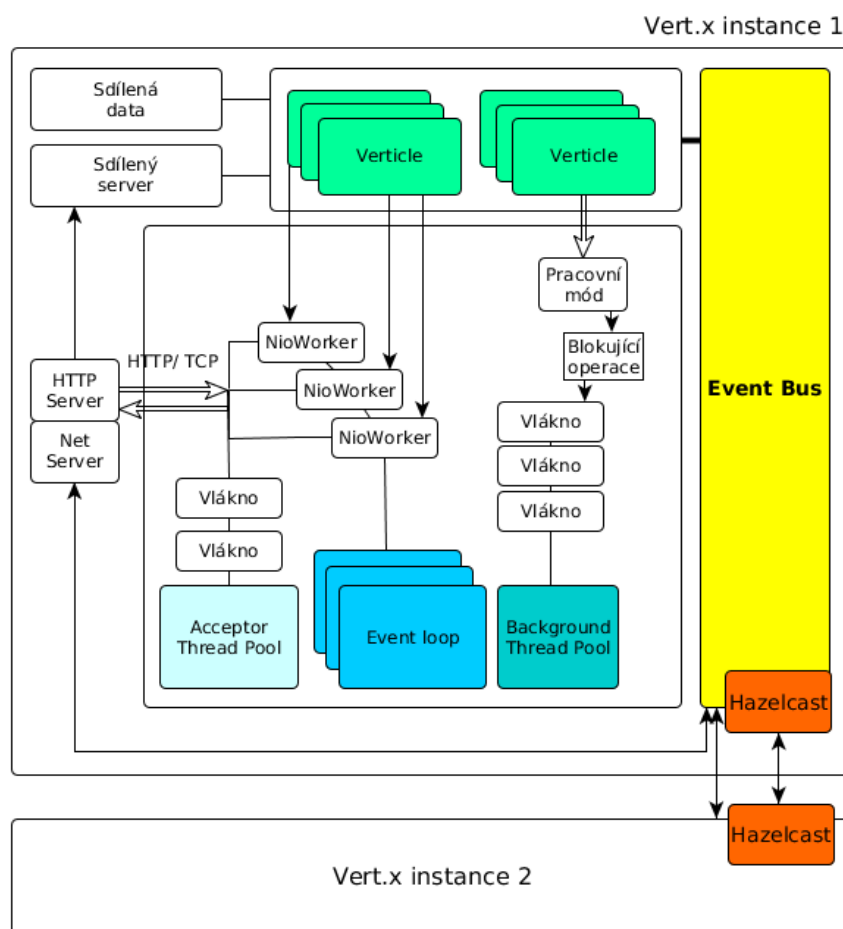
2.1 Historie

Začátek vývoje projektu Vert.x je datován do roku 2011. Tedy rok poté co spatřil světlo světa framework Node.js. Vert.x si za pouhý rok, vydobyl své místo u komunity, která si jej velmi oblíbila.

Hlavním autorem platformy je Tim Fox, který v době začátku vývoje platformy pracoval ve společnosti VMWare. Tato společnost si vzápětí nárokovala všechny zásluhy Tima Foxe na Vert.x platformu. Právníci společnosti vydaly výzvu, ve které požadovali mimo jiné doménu, veškerý zdrojový kód a účet Tima Foxe na Githubu. Z toho důvodu Tim Fox odešel od společnosti v roce 2012. V témže roce projevila o platformu zájem firma RedHat, která nabídla Timovi pracovní místo, absolutně volnou ruku ve vývoji a vedení projektu[1].

Po několika debatách jak s představiteli společnosti RedHat tak i komunitou došel Tim Fox k názoru, že nejlepší pro budoucnost a zdravý rozvoj platformy bude přesu-

¹C10K problém řeší otázku: „Jak je možné obsloužit deset tisíc klientů za pomoci jednoho serveru, a to s co možná nejnižším zatížením serveru



Obrázek 2.1: Architektura Vert.x převzato a upraveno z [10]

nutí celé platformy pod nadací Eclipse Foundation, k čemuž došlo na konci roku 2013. V dnešní době se platforma těší velkému vývoji, který čítá desítky pravidelných příspěvatelů mezi něž patří mimo Tima například Norman Maurer, který se řadí mezi přední inženýry vyvíjející framework Netty.io² a zodpovídá za integraci Netty.io frameworku do platformy Vert.x[2]. Letos Vert.x vyhrál prestižní cenu "Most Innovative Java Technology" v soutěži JAX Innovation awards[4].

2.2 Architektura

Na obrázku 2.1 jsou znázorněny dvě nezávislé Vert.x instance, které spolu komunikují pomocí zpráv. V levé části je blíže zobrazena jedna Vert.x instance, která bude blíže

²framework pro práci se vstupy a výstupy

rozebrána v následujících kapitolách.

2.2.1 Jádno

Velikost samotného jádra aplikace nepřekračuje 10Mb kódu v jazyce Java. V současné verzi je jádro platformy koherentní, dobře čitelné a poskytuje malé, ale za to stabilní API. Jak je popsáno v kapitole 2.3, Vert.x se nesnaží umět vše, ale specializuje se na určitou oblast. Díky těmto faktům je snadno rozšiřitelný a integrovatelný. Lze jej rozšířit o novou funkčnost pomocí balíčků, které lze naléznout v oficiálním repositáři. Pravděpodobnou inspirací byl již zmíněný Node.js, který se svým výkonem vydobyl své místo. Samotnou inspirací byly podle Tima Foxe Node.js a ERLang[2].

Zásadní technologie, které integruje Vert.x.

Netty.io framework pro práci se vstupy a výstupy

Hazelcast In-memory data grid[9]

Netty.io samotný, lze použít pro vývoj webových aplikací stejně dobře jako kterýkoliv jiný nástroj. Jeho specializací však je práce se vstupy a výstupy tzv. IO. V této oblasti poskytuje nízkoúrovňové API, nad kterým Vert.x přidává vyšší míru abstrakce. Druhou technologií, která je pro Vert.x klíčová je popsána v samostatné kapitole 2.2.4.

2.2.2 Asynchronní model

Událostmi řízené programování je podle Tomáše Pitnera[11] základním principem tvorby aplikací s GUI. Netýká se však pouze GUI, je to obecnější pojem označující typ asynchronního programování, kdy je tok programu řízen událostmi na které navěšuje tzv. event handlers³. Typy událostí na které lze reagovat:

- Událost přes GUI
- Chyba
- Zatížení

Události nastávají obvykle určitou uživatelskou akcí(klik či pohyb myši, stisk tlačítka). Událostmi řízené aplikace musí být většinou programovány jako vícevláknové (i když spouštění vláken obvykle explicitně programovat nemusíme). Asynchronní někdy také paralelní model je přímo závislý na způsobu implementace samotným programovacím jazykem. Základním pojmem je zde proces, který je vnímán jako jedna instance programu, který je plánován pro nezávislé vykonávání. Naproti tomu Vlákno⁴ je

³obslužná rutina události

⁴Označuje v informatice odlehčený proces, pomocí něhož se snižuje režie operačního systému při změně kontextu, které je nutné pro zajištění multitaskingu

posloupnost po sobě jdoucích událostí. V dřívější době nebylo potřeba rozlišovat proces a vlákno, protože proces se dále v aplikaci nedělil. Vytvoření vlákna je poměrně drahá a pomalá operace. Což se často obchází vytvořením zásoby uspaných vláken dopředu s nějakým managementem, co vlákna přidává a ubírá dle potřeby. Základním principem Vert.x a jemu podobných frameworků je jedno hlavní vlákno, obvykle pro každý procesor jedno. Takovéto vlákno si pak samo řídí vytváření a přidělování vláken.

Tento model bývá často kritizován, že nutí programátory psát špatně udržitelný kód, především pak v situacích, kdy je potřeba koordinovat výsledky mezi více handlers. Pro tyto situace ovšem vznikla řada nástrojů, které se liší podle použitého jazyka.

Samotné jádro Vert.x je implementováno v jazyce Java a pro Vert.x je tedy důležité, jak moc je dobrá implementace paralelního modelu v tomto jazyce. Neznamena to však, že se celá aplikace musí implementovat výhradně v jazyce Java. Jedinému požadavek pro běh Vert.x instancí je přítomnost Java development Kitu ve verzi 1.7 a novější. Tato verze přinesla nespočet vylepšení, pro jejichž výpis zde není místo. Došlo také na přepsání či úpravy v několika zásadních třídách z balíčku `java.util.concurrent`, což je třída zabývající se prací s multitaskingem a konkurencí.

ExecutorService z balíčku `java.util.concurrent`

CyclicBarrier⁵ z balíčku `java.util.concurrent`

CountDownLatch z balíčku `java.util.concurrent`

File z balíčku `java.nio`

Vylepšený ClassLoader⁶ lepší odolnost vůči deadlockům⁷

Více o `java.concurrent`[8]

Ed Gardoh v roce 2011 ve svém jednoduchém testu[5] prověřil práci s paralelizací úkonů. Z jeho testů vyplývá, že Java 1.7 je až o 40% rychlejší při práci s vlákny nejenom díky nové metodě `Fork/Join`[7].

Návrhový vzor Reactor

„Tradiční“ aplikační serverové kontainery rezervují jedno vlákno pro každou I/O operaci. To v praxi znamená jedno vlákno pro jedno spojení, jak je vidět z obrázku 2.2. takové vlákno pak čeká na odpověď každého řádku volání a prakticky využívá zdroje, které k ničemu nepotřebuje což je velice neefektivní přístup. Vert.x využívá návrhového vzoru *Reactor*, který implementuje Netty.io. Vert.x tuto knihovnu využívá jako

⁷ je odborný výraz pro situaci, kdy úspěšné dokončení první akce je podmíněno předchozím dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce.



Obrázek 2.2: Blokující přístup *převzato a upraveno z [3]*

vestavěný systém. Jeden z hlavních vývojařů Netty.io, Norman Maurer a autor knihy *Netty in action*[3] pracuje na integraci s Vert.x platformou. Z obrázku 2.3 je patrné, že vzor spočívá v běhu jednoho vlákna, kterému tzv. „Selector“ podává požadavky.

Event loop

Event loop využívá Netty.io NioWorkery. Každý verticle⁸ má vlastního NioWorkera, díky tomu je zaručeno, že každé verticle je jedno vláknové. Základem asynchronního modelu je vlákno, které se stará o všechny události. Když událost dorazí vlákno se postará o to aby byla zavolána ta správná obslužná rutina. Každá Vert.x instance interně obsluhuje malý počet těchto vláken, zpravidla pak jedno na každé procesorové jádro. Těmto vláknům se ve Vert.x komunitě říká *Event Loop*. V komunitách Nginx nebo Node.js se ovšem setkáme spíše s pojmem *Run Loop*. Přeloženo volně do češtiny pak „událostní smyčka“. Obrovská nevýhoda takového přístupu je, že nikdy nesmí dojít k blokování hlavního vlákna. Jakmile k tomu dojde, celá aplikace tzv. zamrzne. Při startu verticlu 2.2.3 je pak vybrán jeden event loop, který ho obsluhuje po celý životní cyklus. Event loop je schopný obsloužit tisíce verticlů v ten samý čas. Příklady blokujících volání:

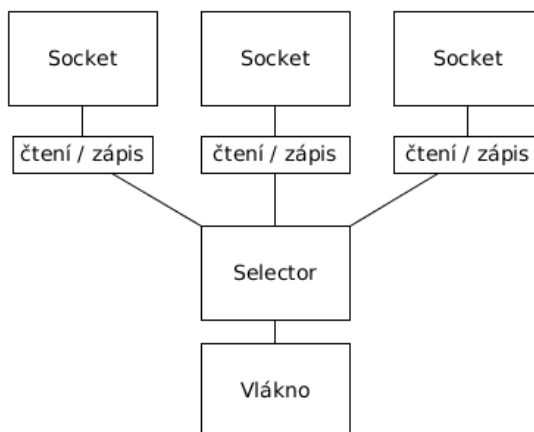
- tradiční API (JDBC, externí systémy)
- dlouhotrvající operace (generování apod.)

Návrhový vzor Multi-reactor

Základ jádra je postaven na tzv. Multi-reactor pattern[13], který vychází z Reactor patternu[6], ten lze charakterizovat několika body:

- aplikace je řízena událostmi

⁸nejmenší jednotka Vert.x více v kapitole Terminologie 2.2.3



Obrázek 2.3: Neblokující přístup pomocí Netty.io převzato a upraveno z [3]

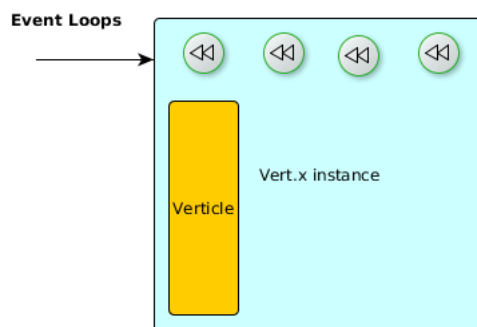
- na události se registrují handlers
- vlákno zpracovává události a spouští registrované handlers
- toto vlákno nesmí být blokováno⁹

Multi-reactor pattern[13] se od Reactor patternu liší pouze tím, že může mít více hlavních vláken. Tím přináší Vert.x možnost pohodlně škálovat instance na více procesorových jader. Jak je vidět z obrázku 2.4 na následující straně Vert.x platforma poskytuje více hlavních vláken, zpravidla však jedno hlavní vlákno na jeden procesor. Toho lze snadno docílit pomocí `Runtime.getRuntime().availableProcessors()` o kterém se dozvíte více v kapitole 3.6. Na obrázku 2.5 na straně 11 pak lze vidět situaci čtyř hlavních vláken na čtyři procesorové jádra.

Hybridní model vláken

Platforma Vert.x přišla s inovací v oblasti hlavních vláken a to takovou, že k hlavním *Event loops* přidala další sadu vláken *Background thread pool*, které jsou vyčleněny z hlavní architektury a poskytující samostatnou kapitolu pro škálování aplikace. To lze ostatně vidět na obrázku 2.1 na straně 4. Díky tomu, lze psát specializované moduly nebo verticle tzv. *workery* pro blokující volání či dlouhotrvající operace aniž by nějak omezovaly běh celé aplikace. Více o *workerech* v kapitole 2.2.3.

⁹pokud dojde k zablokování hlavního vlákna dojde k zablokování celé aplikace např. `Thread.sleep()`, a další z `java.util.concurrent`



Obrázek 2.4: Vert.x instance

2.2.3 Terminologie

Vert.x definuje svou vlastní terminologii, která je specifická jen pro tuhle platformu. Před dalším výkladem je tak nutné porozumět jednotlivým pojmům, které budou vysvětleny v následujících podkapitolách.

Verticle

Základní jednotka vývoje a nasazení. verticle si lze představit jako kus kódu nebo třídu s hlavní metodou. Verticle je tak nejmenší funkční jednotkou Vert.x. Verticle lze spouštět samostatně přímo z příkazové řádky podobně jako skript. Každý verticle běží ve vlastním vlákně z čehož plynou výhody, ale také nevýhody. Vzhledem k tomu, že každý verticle má svůj vlastní classloader nemůže tak sdílet statické metody ani hodnoty proměnných s ostatními. Naopak výhodou je, že programátorovi odpadnou starosti s nejrůznější synchronizací vláken či zámky nad proměnnými. V případě jedno vláknového modelu také odpadají nepříjemné deadlocky. Výzvou je pak sdílení dat mezi jednotlivými komponentami aplikace, které lze v případě Vert.x dvěma způsoby:

- pomocí Message Queue[15] dále jen MQ
- SharedData object a SharedSet *vertx.sharedData()*

Objekty v SharedData musí být immutable¹⁰. V dnešní době je řada MQ frameworků přes které lze vést komunikaci u platformy Vert.x však není potřeba externí služba, protože má vlastní Event Bus o kterém pojednává kapitola 2.2.4. Na obrázku 2.4 je vidět jeden verticle v kontextu jedné Vert.x instance. Následuje sumarizace vlastností verticl:

- nejmenší spustitelná jednotka

¹⁰jakmile jednou takovýto objekt vznikne nejde dále měnit jeho proměnné

- třída / skript
- vykonává neblokující operace
- běží vždy v jednom vlákně
- přímý přístup k API, registrace handlerů, nasazení dalších verticlů

Worker verticle

V standardním verticlu by nemělo nikdy dojít k blokování hlavního vlákna. V dnešní době se bez klasického synchronního volání pravděpodobně neobejdeme, protože většina knihoven a modulů je napsána jako blokující kód. Z toho důvodu je v platformě Vert.x možnost označit verticle jako workera respektive pustit jej jako worker verticle metodou `deployWorkerVerticle` namísto `deployVerticle`. Tím dojde k vyčlenění verticle z asociace na hlavní vlákna a takovému vlákně pak bude přiděleno vlákno z Background thread poolu. Uvnitř takového verticle lze pak vykonávat blokující volání bez blokování celé aplikace. To v praxi rozšířilo možnosti uplatnění celé platformy. Bohužel tímto ztrácíme efektivní možnost škálování pro velký počet konkurenčních vláken je tedy vhodné situovat náročné výpočty na speciálně vyčleně servery. Velikost background thread poolu ve výchozím nastavení čítá 20 vláken. Jak tento počet změnit je popsáno v kapitole 3.6.3.

Vert.x instance

Každý verticle běží uvnitř Vert.x instance 2.4 na předchozí straně a každá instance běží ve vlastní JVM¹¹ instanci. V jedné Vert.x instanci může najednou běžet nespočet Verticlů. Z obrázku 2.7 je patrné, že se jedná o vertikální druh škálování. Kdy na jednom serveru běžíme více Vert.x instancí. Počet instancí je zpravidla roven počtu procesorů, větší počet má zpravidla negativní dopady na rychlost běhu aplikace.

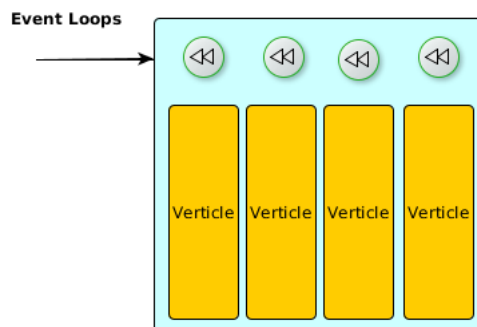
Moduly

Moduly poskytují možnost zapouzdření a znovupoužitelnost funkcionality. V praxi se mohou moduly skládat z více modulů či verticlů ve více programovacích jazycích. Modly mohou být uloženy v centrálním repozitáři¹² nebo může být využit jakýkoliv jiný repozitář. Repozitáře v kterých hledá Vert.x při startu instance dostupné moduly lze definovat v souboru `repos.txt` nacházející se v kořenové složce Vert.x. Každý modul musí mít svůj deskriptor ve formátu JSON¹³. Ukázka deskriptoru je v kapitole 3.4.

¹¹Java Virtual Machine

¹²<http://modulereg.vertx.io/>

¹³je odlehčený formát pro výměnu dat



Obrázek 2.5: Příklad vertikálního škálování `vertx run HelloWorld -instances 4`

Výhody plynoucí z použití modulů:

- classpath¹⁴ je zapouzdřený a díky tomu lze moduly používat mnohem snáze
- všechny závislosti jsou zapouzdřeny v jediném souboru ve formátu ZIP nebo JAR¹⁵
- moduly mohou být umístěny v repozitářích
- Vert.x dokáže automaticky stahovat moduly, pokud je nenalezne v lokální instalaci

Typy modulů lze rozdělit do dvou základních skupin, které lze dále rozdělit podle typu určení modulu.

spustitelné mají definovanou hlavní verticle v deskriptoru, takovéto moduly je pak možné spustit jako samostatné jednotky pomocí parametru `runmod` nebo programově `deployModule`

nespustitelné modul nemá specifikovaný hlavní verticle a lze jej použít v jiném modulu metody `includes`

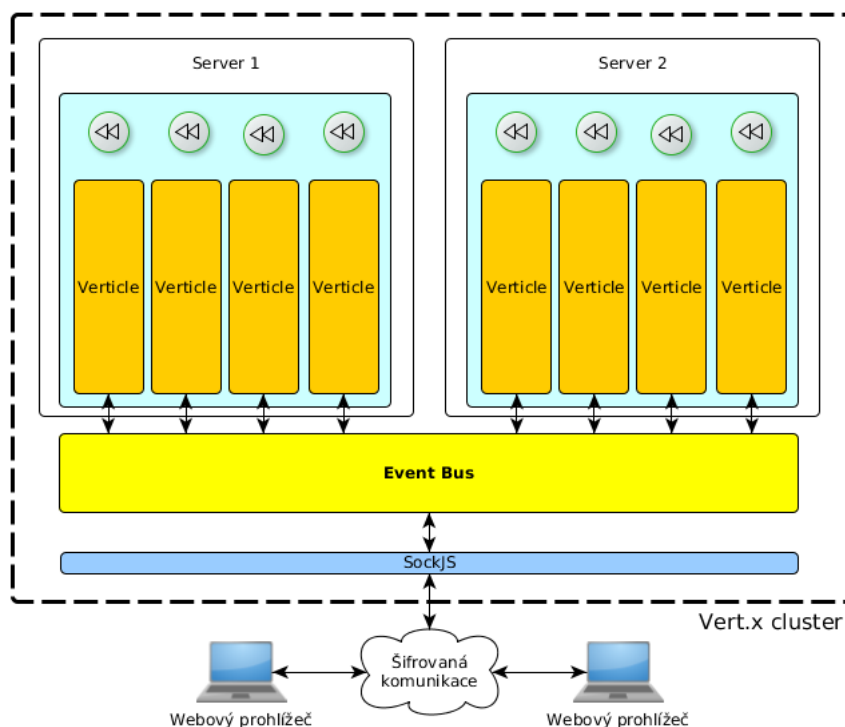
Pro vyčlenění modulu z asociace na event loop stačí přidat parametr `worker: true` do deskriptoru modulu.

2.2.4 Event Bus

Nervový systém celého Vert.x, jehož název lze volně přeložit jako sběrnice událostí. Cílem EventBusu je zpožděkováání komunikace mezi jednotlivými komponentami

¹⁴říká JVM, kde má hledat třídy a balíčky

¹⁵java formát založený na formátu ZIP



Obrázek 2.6: Event Bus distribuovaný mezi dva servery

a vláknů aplikace. Podobně jako při použití externí MQ. Díky faktu, že komponenta Event Bus je implementována přímo v jádru platformy odpadá nutnost používat další knihovny pro práci s MQ a v neposlední řadě také režijní náklady či výpočetní výkon. Jak je vidět na obrázku, komponenta Event Bus je distribuovaná přes všechny instance v clusteru. Samozřejmostí je pak možnost přemostění této komunikace do webového prohlížeče což je detailněji popsáno v kapitole 3.3. Event bus nepřináší takové možnosti v porovnání s profesionálními MQ systémy, jako například transakční zpracování, ale již existuje modul pro přemostění komunikace například do RabbitMQ¹⁶. S vývojáři z týmu vyvíjející RabbitMQ je Tim Fox v úzkém kontaktu[2]. Základní typy komunikace:

- Point to Point
- Publish/Subscribe

typy zpráv:

- String
- primitivní typy (int, long, short, float double, ..)

¹⁶MQ napsaná v Erlangu

- `org.vertx.java.core.json.JsonObject`
- `org.vertx.java.core.buffer.Buffer`

Toto je výčet pouze základních typů zpráv, které Vert.x podporuje v jádře. Není ale vůbec problém výčet stávajících typů rozšířit implementací vlastního modulu. Například modul `bson.vertx.eventbus`¹⁷ rozšíří `EventBus` o možnost používat mnohem komplexnější typy zpráv jejichž výčet se nachází níže.

- `java.util.UUID`
- `java.util.List`
- `java.util.Map`
- `java.util.Date`
- `java.util.regex.Pattern`
- `java.sql.Timestamp`

Mezi doporučené se ovšem řadí JSON, protože je jednoduše serializovatelný mezi jednotlivými programovacími jazyky.

Hazelcast

Jednou z nejdůležitějších architektonických součástí Vert.x je knihovna Hazelcast, kterou tvoří jenom neuvěřitelných 3,1MB kódu v jazyce Java. V platformě Vert.x zaujímá důležité postavení jako In-memory data grid jehož vlastnosti [9] lze podle Ki Sun Song sumarizovat:

- Data jsou distribuovaná a uložena na více serverech ve více geografických lokalitách
- Datový model je většinou objektově orientovaný a ne-relační
- Každý server pracuje v aktivním režimu
- Dle potřeby lze přidávat a odebírat servery

Hazelcast je tedy typ distribuovaného úložiště, které běží jako vestavěný systém, lze díky němu distribuovat celou aplikaci a zasílat zprávy mezi jednotlivými komponentami. Vert.x API využívá Hazelcast API a odstiňuje tak programátora od poměrně nízko

¹⁷<https://github.com/pmlopes/mod-bson-io>

úrovně API Hazelcastu. Když je Vert.x spuštěn, Hazelcast je spuštěn v módu vestavěného systému. Jako nejčastější příklad užití samotného Hazelcastu bývá uváděno ukládání uživatelské session[14]. Hazelcast tedy usnadní práci v situaci, kdy budeme potřebovat uložit uživatelskou session například pro eshop. Mohli bychom využít externí RDBMS tedy databázový server, který by obstarával komunikaci s klienty a udržoval integritu dat díky, kterému by jsme dosáhli stejného výsledku. S využitím knihovny Hazelcast ovšem odpadá nezbytná režie a monitoring, nemluvě o serverových prostředcích. Knihovnu Hazelcast lze využít v několika rolích:

- NoSQL databáze v paměti
- Cache¹⁸
- Data grid
- Zasílání zpráv
- Aplikační škálování
- Clustrování aplikací

2.3 API

Vert.x poskytuje malou sadu metod, kterou lze volat na přímo z jednotlivých verticlů. Funkcionalitu platformy lze jednoduše rozšířit pomocí modulů, které po zveřejnění do centrálního repozitáře může využívat kdokoli a pomáhá tak znovu použitelnosti kódu. Samotné jádro Vert.x je tak velice malé a kompaktní. Vert.x API je rozděleno na *Základní API* a *Kontainer API*.

2.3.1 Základní API

Oproti většině nástrojů se může na první pohled zdát, že je Vert.x API strohé. Při hlubším zkoumání se ukáže pravý opak.

- TCP/SSL server/klient
- HTTP/HTTPS server/klient
- Websockets server/klient, SockJS
- Distribuovaný Event Bus

¹⁸specializovaný typ paměti pro krátkodobé ukládání

- Časovače
- Práce s buffery
- Přístup k souborovému systému
- Přístup ke konfiguraci
- DNS Klient

2.3.2 Kontainer API

Díky této části API může programátor řídit spouštění a vypínání nových modulů a verticlů za běhu aplikace. V praxi jsme tak schopní škálovat aplikaci za běhu či měnit funkcionalitu celé aplikace aniž by to někdo mohl zaregistrovat. Tuto API můžeme také volat přímo z příkazové řádky dále jen CLI¹⁹.

- Nasazení a zrušení nasazení verticlů
- Nasazení a zrušení nasazení Modulů
- Získání konfigurace jednotlivých verticlů
- Logování

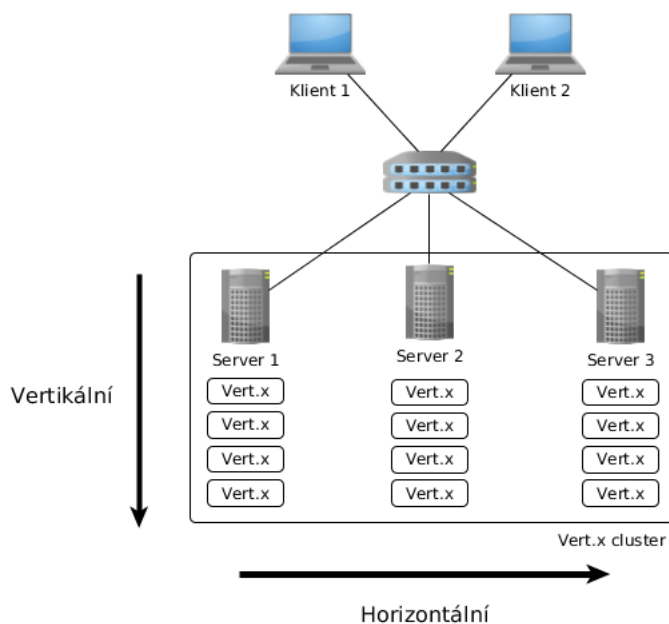
2.3.3 Polyglot

Polyglot je označován člověk, který ovládá více jazyků. V terminologii Vert.x to znamená, že API je dostupná ve více programovacích jazycích. Což v praxi znamená, že si programátor může sám zvolit v jakém jazyce bude implementovat svůj kód. Díky faktu, že spolu všechny verticly komunikují skrze zprávy je tak možné mít část aplikace napsanou například v jazyce Java a druhou část v jazyce Python. Tento fakt hodně pomáhá celé platformě nalákat větší množství nových programátorů. Takový vývojář klientské části aplikace většinou píše v jazyce JavaScript, naproti tomu vývojář serverové části určitě píše spíše v jazyce Java nebo Scala²⁰. Výčet podporovaných jazyků ve verzi 2.0. Do verze 3.0 se pak chystá automatické generování API pro každý jazyk.

- Java, Scala, Groovy
- Javascript, CoffeeScript
- Ruby

¹⁹Command Line Interface

²⁰programovací jazyk, který běží na JVM



Obrázek 2.7: Horizontální a vertikální škálování

- Python
- PHP
- Clojure

2.4 Clustering

Díky integraci Hazelcastu získala platforma Vert.x řadu zajímavých vlastností mezi které patří také možnost horizontálního škálování. Jak je vidět z obrázku 2.7, jde o typ škálování do šířky, propojováním více serverových instancí dohromady. Na těchto instancích pak běží Vert.x platforma respektive Hazelcast, který spojuje všechny klienty dohromady. To v praxi znamená, že můžeme aplikaci jednoduše škálovat přes více serverů bez nutnosti běhu dalších služeb a režijních nákladů. Přímou za běhu aplikace lze přidávat další Vert.x instance do clusterů. Samotná konfigurace clusteru není pak nic složitějšího a odehrává se v souboru `conf/cluster.xml` a spočívá v nastavení členů clusteru nebo specifikování multicastové²¹ adresy a portu na které bude Hazelcast po startu vyhledávat členy clusteru. Pro spuštění aplikace v režimu cluster ji stačí spustit s parametrem `-cluster`. Pokud se na daném serveru nachází více síťových rozhraní je potřeba

²¹logický identifikátor skupiny síťových hostů

specifikovat *-cluster-host*. Na tomto rozhraní pak bude komunikovat Hazelcast. V kapitole 3.6.2 je pak tato možnost využita pro běh Vert.x clusteru na odlišném rozhraní než běží webový server. Velkou nevýhodou je pak nemožnost nasadit cluster na veřejné síti. V současné době Event Bus nepodporuje šifrované zprávy a jediná možnost jak toho dosáhnout, je pomocí nejrůznějších privátních tunelů. Ve verzi 3.0 má být plně implementována šifrovaná komunikace a nebude tak nic bránit nasazení na veřejné síti.

2.4.1 Vysoká dostupnost

Samostatnou kapitolou v oblasti clusteringu je HA²² česky tedy vysoká dostupnost. Díky Hazelcastu ji lze řešit již na aplikační úrovni, a není potřeba dalších služeb, které řeší vysokou dostupnost.

Automatické zotavení z havárie

Pokud je modul spuštěn s argumentem *-ha* a dojde k pádu Vert.x instance. Modul bude automaticky nasazen na jiné instanci v clusteru. V takovém případě již není potřeba spouštět modul s parametrem *-cluster*. Jak je vidět na obrázku 2.8 v případě pádu Serveru 2, tedy části aplikace, která komunikuje s databází dojde automaticky k novému nasazení této části do nové instance, která je taky členem Vert.x clusteru. Výpadek tak bude pro uživatele skoro nepostřehnutelný.

HA skupiny

V případě spuštění modulů v režimu HA lze pak specifikovat logické skupiny. Pro spuštění instance v určité HA skupině stačí přidat parametr *-hagroup <jméno skupiny>*. Díky tomu lze přesně určit, kde se mají moduly v případě pádu nasadit. To je v praxi vhodné především v situacích, kdy je do internetu vystavena pouze část clusteru. Například jako na obrázku 3.8.

Kvorum

Při spuštění Vert.x instance lze specifikovat kvorum²³. Pokud nebude splněno kvorum nebude instance nasazena v režimu HA. Kvorum lze pak snadno spočítat ze vzorce $Q = 1 + N/2$, kde N je počet serverů. Pokud dojde při běhu aplikace k porušení kvora bude režim HA automaticky vysazen.

²²HA - High Availability

²³minimální počet serverů pro zajištění vysoké dostupnosti



Obrázek 2.8: Clustering mezi třemi Vert.x instancemi

Vertigo cluster

Důkazem, jak dobrá je architektura je rozšíření samotné funkcionality clusteru. Modul Vertigo[19], obohacuje platformu o značné možnosti v oblasti clusterování. Jak píše sám autor projektu „*Verigo kombinuje koncept real-time systémů a Flow-based programování.*“. Architektura Vertigo by vydala sama na celou bakalářskou práci. Proto jen ve stručnosti Vertigo podporuje dávkování zpráv mezi komponentami. Umožňuje vytváření a monitorování sítí mezi clustery a podporuje živou změnu bez restartu těchto sítí. Usnadňuje distribuci modulů přes celý cluster pro snadné nasazení v kterémkoliv jeho koutě z kteréhokoliv jeho místa.

2.5 Porovnání s Node.js

V následující kapitole bude porovnána platforma Vert.x s již zmíněnou platformou Node.js. Výkonnostní test[17] je převzat od samotného autora projektu a jsou v něm zahrnuty jazyky, které v té době platforma podporovala. V druhé části kapitoly 2.5.1 je pak tabulka 2.1 srovnání odezev s vybranými webovými platformami z daty ze zdroje[16].

2.5.1 Výkon

Tato kapitola se zabývá výkonnostními testy jednotlivých platforem. V prvním testu je obsaženo více programovacích jazyků, ve kterých byla implementována stejná logika pod platformou Vert.x. Rychlost aplikace implementované v jiném jazyce než Javě je pak závislá na konkrétním adaptéru.

Metody testování

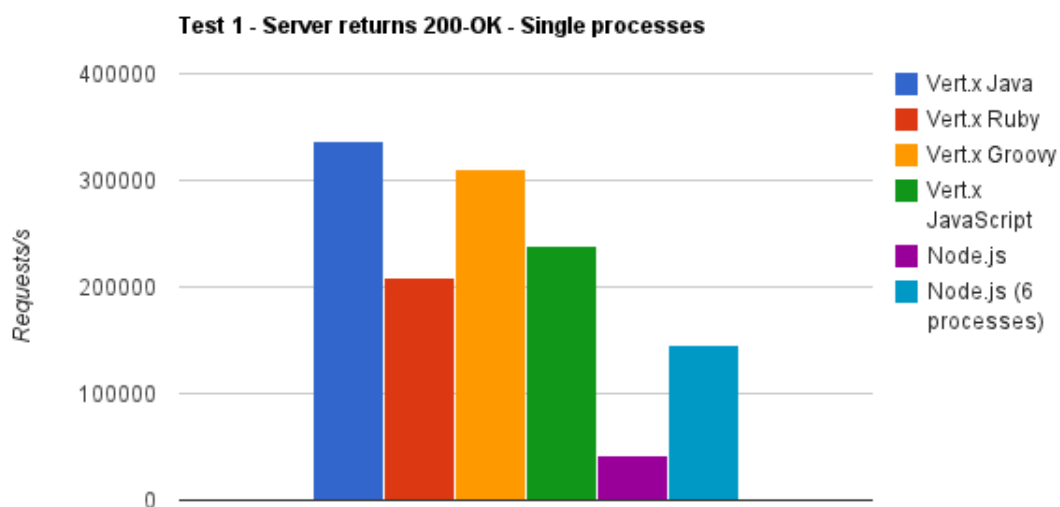
V obou testech je testovaná aplikace škálovaná na šest procesorových jader tedy byla spuštěna s parametrem *-instances 6*, oproti tomu je spuštěna aplikace Node.js ve dvou variantách. Samostatná a šest procesů v jednom clusteru. V legendě grafů je to odlišeno příponou *cl*.

1. Triviální dotazování serveru a návrat statusu 200²⁴
2. Dotaz na statický soubor o velikosti 72 bytů

Hardware

V prvním testu od Tima Foxe je použit AMD Phenom II X6, 8GB RAM a systém Ubuntu 11.04. Tento procesor se 6 jádry není úplně běžný proto je výklad doplněn o druhý

²⁴HTTP status - OK

Obrázek 2.9: Výsledky prvního testu *Tim Fox* [17]

test, který proběhl na Sandy Bridge Core i7-2600K, 8GB RAM a SSD disku a systému Ubuntu 12.04.

Výsledky

Jak lze vidět na obrázku 2.9 a 2.10 výsledky obou testů lze shrnout do jedné věty. Vert.x zvládá řádově o desítky tisíc více odpovědí než platforma Node.js a to i v případě režimu clusteru.

Srovnání s vybranými platformami

Metoda srovnání s ostatními platformami je založená na podobném principu jako předchozí testy s tím rozdílem, že místo statického souboru vrací odpověď ve formátu JSON. Na straně serveru tedy musí dojít k JSON serializaci.

Platforma	Průměrná odezva	Maximální
Vert.x	1,2ms	18,7ms
Netty	1,3ms	24,0ms
Ruby on Rails	1.8ms	241.6
Node.js	3.7ms	12,5

Tabulka 2.1: Srovnání odezvy, převzato a upraveno z [16]

2.5.2 Vlastnosti

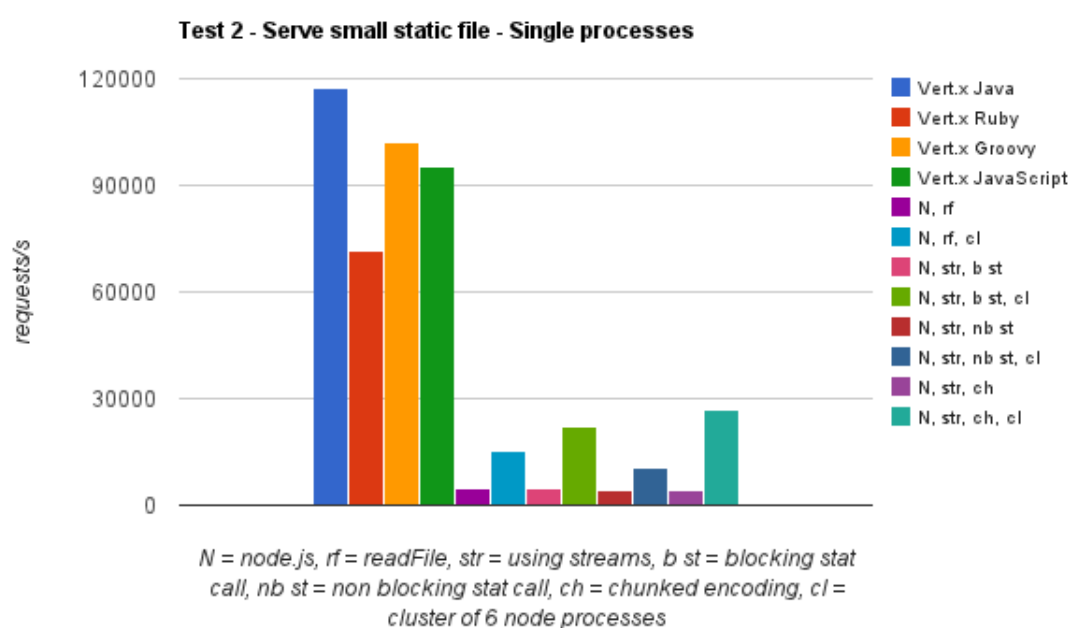
Následující tabulka ukazuje srovnání důležitých vlastností jednotlivých platforem, jejichž důležitost byla popsána v předchozích kapitolách.

Vlastnost	Node.js	Vert.x
CLI	Ano	Ano
Cluster	Ano	Ano
Moduly	Ano	Ano
HA	Ne	Ano
MQ	Ne	Ano
Hybridní model vláken	Ne	Ano
In-memory data grid	Ne	Ano
Polyglot	Ne	Ano

Tabulka 2.2: Srovnání vlastností s Node.js

2.5.3 Závěr srovnání

Výsledkem srovnání je tedy fakt, že pokud by se dnes někdo rozhodoval o výběru platformy pro novou real-time aplikaci měl by určitě zvolit platformu Vert.x, která poskytuje řádově větší výkon a počet vlastností, nehledě na fakt, že v případě Node.js lze psát aplikaci pouze v jazyce JavaScript, který se může jevit jako naprosto nevhodný pro Enterprise aplikaci.



Obrázek 2.10: Výsledek druhého testu *Tim Fox* [17]

3 Praktická část

V této kapitole je popsán software a postupy použité při implementaci a nasazení malé Vert.x aplikace pro správu myšlenkových map.

3.1 Návrh

Aplikace bude složena ze dvou částí. První bude serverová část, která bude pracovat s mapami a bude obsluhovat klienty, kteří budou provádět jednotlivé akce. Druhá část potom bude na straně webového klienta, tedy část realizovaná pomocí D3.js, která bude mít na starosti vykreslování a reagování na uživatelské akce.

3.1.1 Cíle aplikace

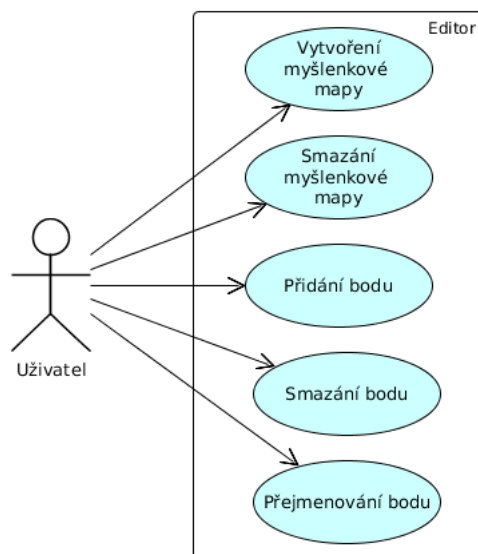
- Přidání a odstranění myšlenkové mapy
- Přidání a odstranění bodu v myšlenkové mapě
- Přejmenování bodu v myšlenkové mapě

3.1.2 Architektura

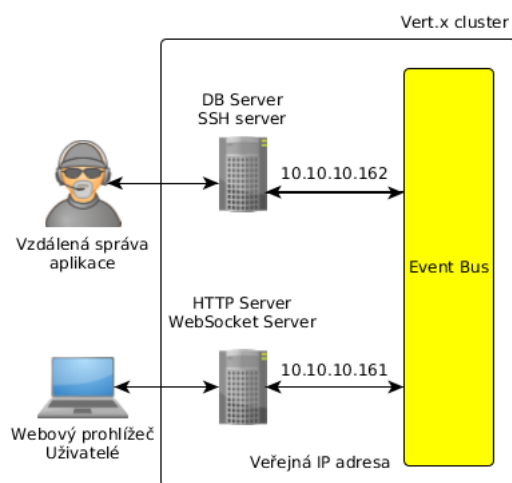
Jak je vidět na obrázku 3.2 klienti se budou připojovat přes jeden webový server, který bude mít otevřené porty 80. S druhým serverem bude spojený na úrovni Hazelcast clusteru. Vzhledem k situaci, která je blíže popsána v kapitole 3.6.2, kdy je webový server připojen do dvou sítí není potřeba šifrování ani nejrůznějších tunelů. Druhý server bude sloužit pro komunikaci s databází a také jako *exporter* myšlenkových map do obrázků ve formátu PNG.

3.2 Vlastní implementace

Vzhledem k rozsahu práce bude v následující kapitole ukázána a popsána většina implementačních částí.



Obrázek 3.1: Případy užití



Obrázek 3.2: Architektura nasazené aplikace

3.2.1 Řídící verticle

Vzhledem k tomu, že bude aplikace nasazená na více serverech ve více rolích bylo by také zapotřebí více specializovaných modulů. Mnohdy je výhodnější implementovat řídící verticle, který bude mít na starost nasazovat moduly, dle dané konfigurace. Část kódu takového startéru vypadá následovně:

```
var container = require('vertx/container');
var console = require('vertx/console');

var config = container.config;

if("webserver" in config) {
  container.deployModule('io.vertx~mod-web-server~2.0.0-final',
    config.webserver, config.webserver.workers, function(err, ID){
    if (err) {
      console.log(err)
    }
  });
}
```

Každá metoda *deploy* má jako poslední parametr obslužnou rutinu pro případ selhání. V mnoha případech se také hodí ID nasazení, díky němuž lze později toto nasazení zrušit.

Samotný kód vytáhne z třídy *container* konfiguraci celého modulu a zeptá se jestli se v něm nenachází daná role. Potom už jen stačí aby daný konfigurační soubor obsahoval klíč *webserver* s danou konfigurací, která se nachází níže. Obdobně je implementováno startování editoru, databázového modulu a exportéru. Spuštění modulu či verticle jde samozřejmě i ručně z příkazové řádky.

Kód 3.1: Spuštění modulu z příkazové řádky

```
vertx runmod io:majklk~mindmapeditor~0.0.1 -conf
  /srv/mindmap/conf/webserver.json -instances 3
```

Spuštěním modulu z příkazové řádky se Vert.x podívá do deskriptoru modulu, v kterém by měl najít cestu k hlavnímu verticlu. Ten následně spustí a předá mu danou konfiguraci. Samotný verticle pak může pracovat s Vert.x instancí.

Kód 3.2: Konfigurace serveru 1

```
{
  "name": "MindMap editor server 1 - HTTP + WebSocket",
  "webserver": {
```



```

    "workers": 3,
    "web_root": "web",
    "host": "10.10.10.161",
    "port": 80,
    "bridge": true,
    "inbound_permitted": [
      { "address": "mindMaps.list" },
      { "address": "mindMaps.save" },
      { "address": "mindMaps.delete" },
      { "address": "mindMaps.exporter.svg2png" },
      { "address_re": "mindMaps\\.editor\\.\\.+" }
    ],
    "outbound_permitted": [
      { "address_re": "mindMaps\\.events\\.\\.+" }
    ]
  }
}

```

Většina jména parametrů mluví sami za sebe, kromě *bridge*, *inbound permitted* a *outbound permitted*. Pokud je první z nich nastaven na hodnotu *true* tak začnou platit pravidla, která jsou nadefinována v *inbound permitted*, *outbound permitted*. Jde o tzv. bridge mezi webovým prohlížečem a Event busem jako takovým. *Inbound permitted* říká jaké adresy se mají pustit dovnitř, analogicky potom *outbound permitted* říká, co může jít ven. Výhodou je pak možnost specifikovat adresu pomocí regulárních výrazů.

Základní adresářová struktura

```

main.js //stará se o spouštění celé aplikace v několika rolích
editor.js //obsluha editoru
database.js //obsluha událostí
web //kořenová složka webové části

```

3.2.2 Editor

Kód 3.3: Zaregistrování obslužné rutiny v jazyce JavaScript

```

var eb = vertx.eventBus;

var mujHandler = function(zprava) {
  console.log('Přišla zpráva ' + zprava);
}

eb.registerHandler('test.tisk', mujHandler);

```

Obdobně to pak vypadá v ostatních jazycích.

Kód 3.4: Zaregistrování obslužné rutiny v jazyce Java

```
EventBus eb = vertx.eventBus();

Handler<Message> mujHandler = new Handler<Message>() {
    public void handle(Message zprava) {
        System.out.println("Přišla zpráva " + zprava.body);
    }
};

eb.registerHandler("test.tisk", mujHandler);
```

Na podobném principu je pak postaven zbytek serverové části. Na jednotlivé události jsou zaregistrovány podobné obslužné rutiny, které většinou dohledají konkrétní mapu a provedou nad ní nějakou změnu. Výslednou mapu uloží a pošlou událost všem klientům jako je vidět na diagramu 3.3.

Kód 3.5: Publikování zprávy v jazyce JavaScript

```
eventBus.publish('editor.udalosti.'+mindMap._id, {"udalost":
    "bodPridan", parentKey: args.parentKey, node: newNode});
```

Díky metodě *publish*, která je samozřejmě dostupná ve všech jazycích můžeme publikovat událost všem klientům, kteří mají zaregistrovanou obslužnou rutinu na identickou adresu, která je unikátní pro každou mapu.

3.2.3 Integrace s databází MongoDB

Vzhledem k jednoduchosti aplikace nebyl vytvořen diagram tříd. V aplikaci budou pouze dva modely a to mapa a její potomek, který se od mapy liší pouze tím, že má místo globálního unikátního identifikátoru pouze unikátní identifikátor v rámci jedné mapy.

V centrálním repozitáři již existuje modul pro komunikaci s MongoDB¹, který poskytuje jednoduchou API pro asynchronní komunikaci s databází. Zaregistrujeme tedy obslužnou rutinu pro uložení myšlenkové mapy. Obdobně pak pro každou akci z definovaných cílů.

Kód 3.6: Uložení myšlenkové mapy do databáze

```
var eventBus = require('vertx/event_bus');
var console = require('vertx/console');
```

¹<https://github.com/vert-x/mod-mongo-persistor>

```

eventBus.registerHandler('mindMaps.save', function(mindMap) {
  dotaz_do_databaze = {action: "save", collection: "mindMaps",
    document: mindMap}
  eventBus.send('vertx.mongopersistor', dotaz_do_databaze,
    function(reply) {
      if (reply.status === "ok") {
        // vše v pořádku
      } else {
        console.log(reply.message);
      }
    });
});

```

Kód 3.7: Konfigurace serveru 2

```

{
  "name": "MindMap editor server 2 databázový modul, obrazkový
    exporter",
  "exporter": {
    "workers": 5
  },
  "mongodb": {
    "address": "vertx.mongopersistor",
    "host": "localhost",
    "port": 27017,
    "db_name": "mindmap_editor",
    "pool_size": 20
  },
  "shell": {
    "crash.auth": "simple",
    "crash.auth.simple.username": "admin",
    "crash.auth.simple.password": "heslo",
    "crash.ssh.port": 2000
  }
}

```

_id unikátní identifikátor²

name název samotné mapy

children potomci

²podtržítkem se běžně označující neměnné proměnné

```

{
  "_id": "1234-6545-5612-3456",
  "name": "Živočichové",
  "children": [
    {
      "key": "1",
      "name": "Obratolvcí",
      "children": [
        {
          "key": "2",
          "name": "Ryby"
        },
        {
          "key": "3",
          "name": "Plazi"
        }
      ]
    }
  ],
  {
    "key": "4",
    "name": "Bezobratlí"
  }
]
}

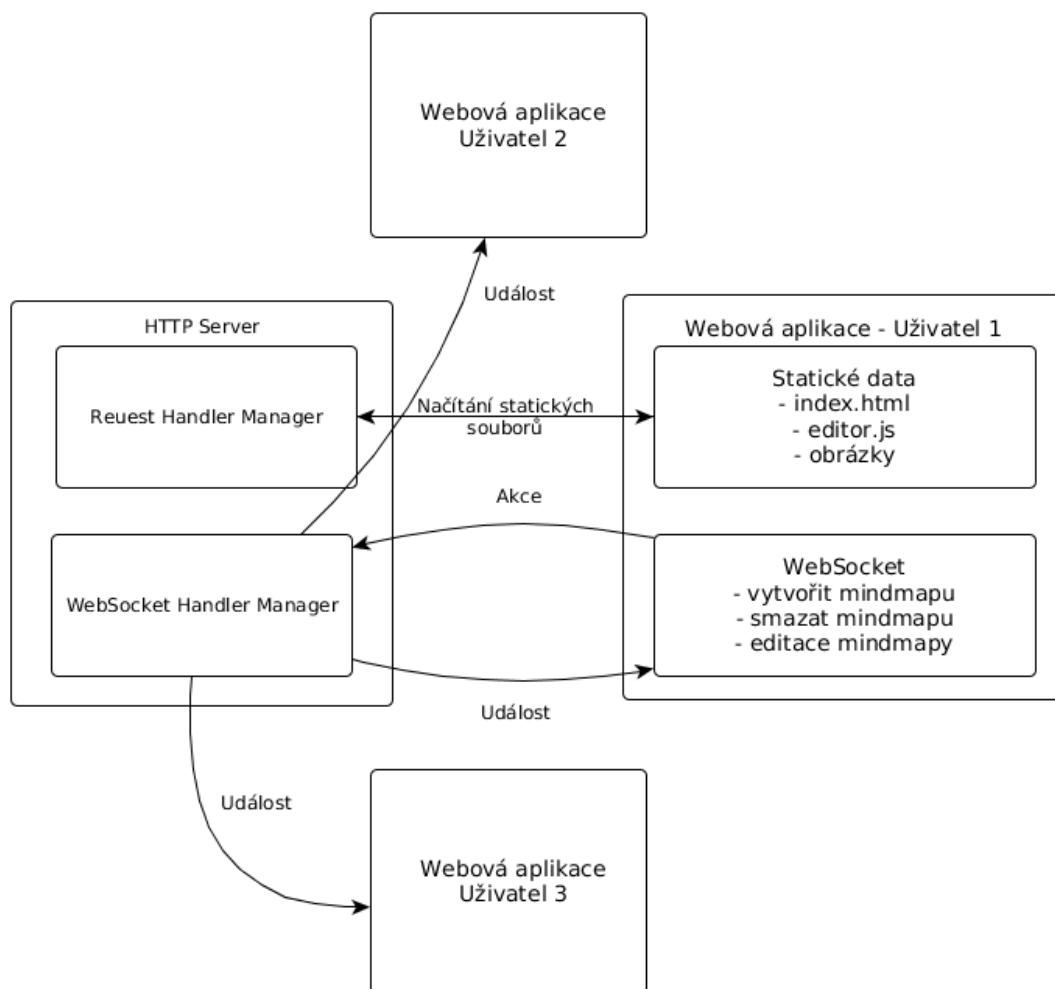
```

3.3 Komunikace v reálném čase

Po načtení myšlenkové mapy přichází na řadu aspekty komunikace v reálném čase. V rámci editoru myšlenkových map budou implementovány tři základní operace.

- Přidání objektu do myšlenkové mapy
- Odstranění objektu z myšlenkové mapy
- Přejmenování objektu v myšlenkové mapě

V tradiční webové aplikaci by to znamenalo implementaci těchto metod typem požadavek-odpověď jako operací konkrétní API. Při přidání objektu by se zavolala API a nazpět by přišla odpověď zda-li byla akce úspěšná. Pokud bychom však chtěli mít editor, který by propagoval změny ke všem, kdo mají myšlenkovou mapu otevřenou museli bychom znát přihlášené uživatele, kterým by server poslal notifikaci o změně. Mnohem jednodušší cesta je rozdělení požadavku a odpovědi do dvou částí což odpovídá návrhovému



Obrázek 3.3: Komunikace v reálném čase

vzoru Command. V takovém případě při otevření webového prohlížeče s danou myšlenkovou mapou dojde k zaregistrování klienta na určitou adresu. V případě jakékoli změny, kterou provede jiný uživatel nebo kdokoliv jiný, dojde k odeslání události všem zaregistrovaným klientům okamžitě v době vykonání události. Tuto situaci lze vidět na obrázku 3.3. V případě, kdy u uživatele dojde k vyvolání akce, ostatním uživatelům bude zaslána událost, která s sebou nese všechny informace o změně. Všem klientům zaregistrovaným na stejnou adresu přijde stejná událost. Tento typ zasílání zpráv je znám jako návrhový vzor Publish/Subscribe.

3.3.1 Akce

Když bude uživatel chtít změnit myšlenkovou mapu (přidat objekt, odebrat objekt nebo přejmenovat objekt), vyšle akci na server. Tato akce bude poslána přes přemostěný event bus, který byl představen v kapitole 2.2.4. Na straně serveru je pak verticle, který má zaregistrovány metody na příchozí akce. Samotná akce nemá žádnou návratovou hodnotu, pokud tak dojde k chybě nedojde k vyslání události, která s sebou nese změny myšlenkové mapy.

3.3.2 Události

Pokud uživatel otevře webový prohlížeč s konkrétní myšlenkovou mapou, dojde tak k přihlášení odběru událostí nad touto myšlenkovou mapou. Pokud ji někdo změní tento uživatel dostane stejnou událost s informací o změně jako každý jiný uživatel přihlášený k odběru událostí. Na straně klienta tak budou implementovány metody, které budou mít definované chování pro každou z definovaných událostí: přidání, odebrání a přejmenování objektu v myšlenkové mapě.

3.3.3 Klientská aplikace

V následující kapitole budou objasněny technologie použité na straně webového prohlížeče.

Struktura webové části

```
index.html
js/client.js //obsluha editoru
js/editor.js //vykreslování grafu - D3.js
js/vertxbus.js //knihovna pro napojení na Event Bus
css //složka pro kaskádové styly
```

Event Bus v prohlížeči

Nejdůležitější částí klientské aplikace je Vert.x vrstva nad SockJS[23], která zabaluje komunikaci se serverem na protokolu WebSocket[24]. Knihovna naváže a udržuje neustálé spojení se serverem. Implementací metod *eb.onopen* a *eb.onclose* může patřičně reagovat na otevření a uzavření spojení. Jediným parametrem samotného EventBusu je pak URL adresa na kterou se má připojit.

Kód 3.8: Připojení Event busu z prohlížeče a inicializace editoru

```
var eb = new vertx.EventBus(window.location.protocol + '//' +
    window.location.hostname + ':' + window.location.port +
```

```

    '/eventbus');

eb.onopen = function() {
    //tato metoda se volá po úspěšném navázání komunikace
    //zde již můžeme posílat zprávy
    eb.send("mindMaps.save", mapa, function(reply){
        eb.send("mindMaps.find", mapa._id, function(mindMap){
            new MindMapEditor(mindMap, eb); //inicializace editoru
        });
    });
}

```

D3.js

Knihovna D3.js napsaná v jazyce JavaScript je velký pomocník při práci s grafy. Díky tisícům již hotových ukázek si stačí jen vybrat, kterou použít. Všechny jsou totiž volně dostupné. Pro vykreslení myšlenkových map tak poslouží Collapsible Tree[20], který se výborně hodí právě pro stromové struktury. Implementace editoru je tak v souboru *editor.js*

Kód 3.9: D3.js inicializace dat

```

var tree = d3.layout.tree()
    .size([vyska, sirka]);

var diagonal = d3.svg.diagonal()
    .projection(function(d) { return [d.y, d.x]; });

d3.json(mindMap);

```

3.4 Polyglot vývoj a moduly

Pro větší znovupoužitelnost je vhodné své aplikace balit do modulů, které pak lze snadno distribuovat v různých formátech. Aby se z editoru stal modul musí být adresářová struktura upravena následovně.

```

mods //složka pro další moduly
lib //složka pro ostatní knihovny např. další jar soubory
main.js //stará se o spouštění celé aplikace v několika rolích
editor.js //obsluha editoru
database.js //obsluha událostí
web //kořenová složka webové části

```

```
mod.json //deskriptor modulu
```

ClassLoader při startu modulu automaticky načte všechny knihovny nacházející se ve složce *lib*. Ručně to lze pak přes parametr *-cp*³. Pokud je modul spustitelný, musí deskriptor modulu obsahovat parametr *main*. Což je cesta a jméno hlavního verticle. Modul, který nemá specifikovaný parametr *main* lze pak snadno použít v jiném modulu pomocí parametru *includes*. Modul se pak musí nacházet ve složce *mods*. Nastavením parametru *worker* na *true* docílíme vyčlenění všech verticlů z asociace na Event loop jak bylo řečeno v 2.2.2. Parametr *auto-redeploy* se hodí především v časech vývoje, při každé změně kódu dojde k novému nasazení modulu.

```
{
  "main": "main.js",
  "worker": true,
  "includes": "io.majklk~mindmapeditor~0.0.1",
  "auto-redeploy": true
}
```

Jmenná konvence modulů vychází ze Java konvence[25] pro pojmenování balíčků ke které přidává akorát číslo verze *com.mycompany my-mod 1.0*. Pro publikování modulu jsou pak důležité nejruznější metadata, které je dobré specifikovat.

```
{
  "developers": ["Michael Kutý"],
  "name": "MindMap Editor",
  "version": "0.0.1",
  "description": "Jednoduchý editor myšlenkových map",
  "keywords": ["vert.x", "mongodb", "d3js"]
}
```

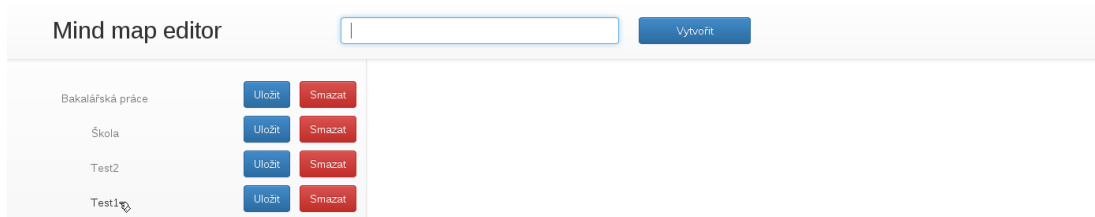
Kód 3.10: Spuštění modulu v jazyce Java

```
container.deployModule("io.majklk~mindmapeditor~0.0.1", JSONconfig);
```

Kód 3.11: Verticle v jazyce Java

```
import org.vertx.java.core.Handler;
import org.vertx.java.core.eventbus.Message;
import org.vertx.java.core.json.JsonObject;
import org.vertx.java.platform.Verticle;
```

³ClassPath



Obrázek 3.4: Webová aplikace

```
//třída musí rozšiřovat třídu Verticle
public class ImageExporter extends Verticle {

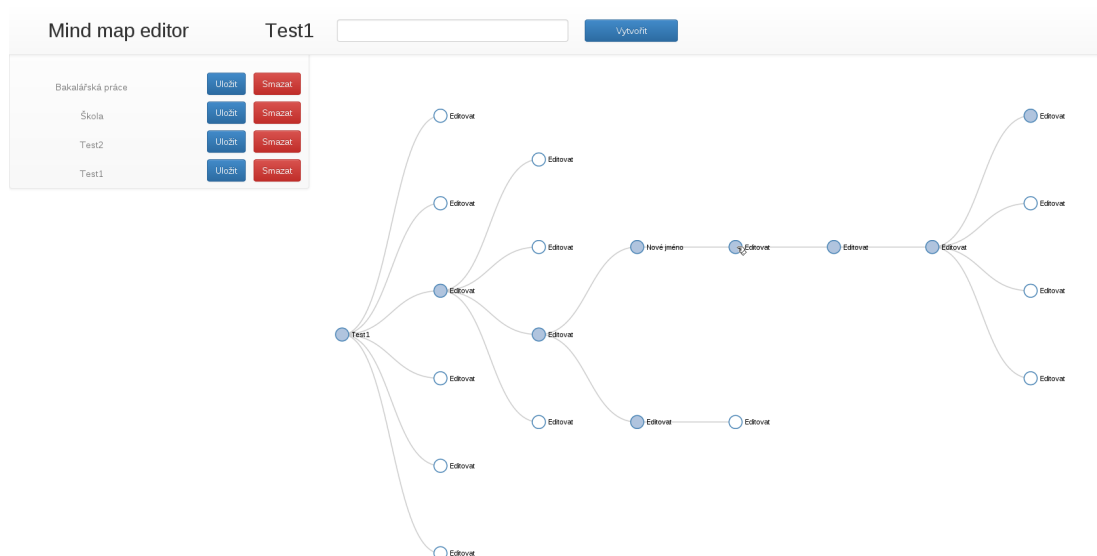
    public void start() {
        Handler<Message<JsonObject>> exportHandler = new
            Handler<Message<JsonObject>>() {
                public void handle(Message<JsonObject> message) {
                    String svg = message.body().getString("svg");
                    String css = message.body().getString("css");
                    //odpověď volajícímu req/res
                    message.reply(new JsonObject().putString("data",
                        getPng(svg, css)));
                }
            };
        //zaregistrování obslužné rutiny
        vertx.eventBus().registerHandler("mindMaps.exporter.svg2png",
            exportHandler);
    }

    private String getPng(String svg, String css) {
        \\tělo metody
    }
}
```

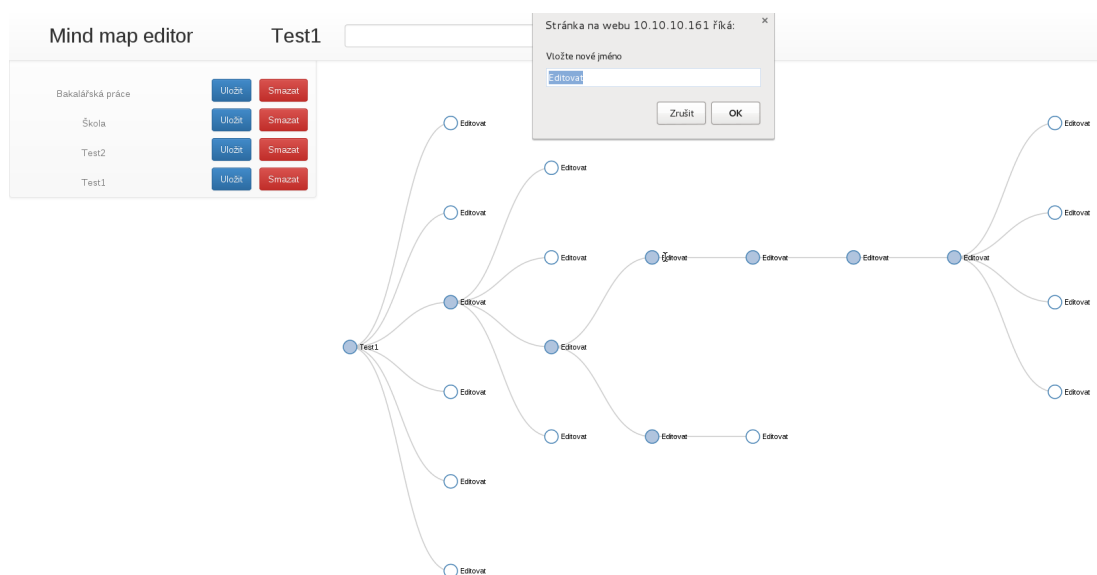
3.5 Základní software

Jádrem serveru je operační systém Ubuntu[21] 14.04 LTS⁴ Server Edition s kódovým označením Trusty Tahr. Je to osvědčený systém, který bude mít podporu do roku 2019.

⁴dlouhodobá podpora



Obrázek 3.5: Webová aplikace - otevření myšlenkové mapy



Obrázek 3.6: Webová aplikace - akce - editace

Systém má aplikaci pro správu softwarových balíčků aptitude. Všechny aplikace kromě samotného Vert.x šli hravě nainstalovat.

3.5.1 Java

Jako hlavní přísadou celého prostředí je otevřená implementace Java Platform, knihovna OpenJDK ve verzi 7.

3.5.2 Vert.x

Jediná služba, která se zatím nenachází jako systémový balíček je samotný Vert.x. Pro jeho instalaci je nutné stáhnout distribuci ze stránek platformy. Tento archiv potom rozbalit do požadované lokace. Následně stačí v závislosti na konkrétním systému přidat soubor *vertx/bin/vertx* do systémové proměnné *PATH*. Poté by měla být funkční interakce s platformou pomocí příkazové řádky. Příklad proměnné *PATH* lze vidět v kapitole 3.5.4. Správné nastavení lze otestovat napsáním *vertx* do příkazové řádky. Správný výstup jsou pak pomocné informace pro komunikaci s platformou tzv. help.

3.5.3 Databázový server

Pro ukládání myšlenkových map je použita NoSQL databáze MongoDB ve verzi 2.6. MongoDB má za sebou více než pět let vývoje a několik obřích investic[22] do dalšího vývoje. V dnešní době existuje nespočet NoSQL databází, vzhledem k tomu, že již existuje Vert.x modul pro pohodlnou asynchronní spolupráci s touto databází byla vybrána právě tato NoSQL databáze. Pro instalaci stačí využít balíčkovací systém aptitude. Pro samotné nastavení databáze, zde není prostor a postačí tak výchozí stav, kdy databáze naslouchá na portu 27017 což lze vidět i z konfigurace aplikace v ukázce kódu 3.13

```
apt-get install mongodb-server -y
```

3.5.4 Nasazení produkční služby

V současné verzi(2.1.2) Vert.x nepodporuje běh v režimu daemon⁵. Nasazení v režimu daemon je však nutnost pro běh v produkčním prostředí. Pro běh aplikace MindMap editoru byla využita systémová služba Supervisor⁶, která běží jako linuxový daemon a stará se o běh aplikace, v případě pádu se ji pokusí znovu nasadit. Samotná konfigurace služby pro běh v Supervisoru pak obsahuje základní parametry. Ve verzi 3.0 je však plánovaná funkce běhu v režimu daemona a nebude tak tato berlička potřeba.

⁵je program, který běží v pozadí, čeká na události, které nastanou, reaguje na ně a poskytuje služby.

⁶supervisord.org

Kód 3.12: Konfigurace produkční služby

```
[program:vertx_mindmap_editor]
directory=/srv/mindmap/app
environment=PATH="/srv/vert.x-2.1RC3/bin/vertx"
environment=JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64/"
command=vertx runmod io.majklk~mindmapeditor~0.0.1 -conf
    /srv/mindmap/conf/allinone.json
user=root
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=/srv/mindmap/app/app.log
stderr_logfile=/srv/mindmap/app/error.log
startsecs=10
stopwaitsecs=600
```

Nejdůležitější parametry jsou *environment* a *command*, které umožňují pohodlně spustit libovolnou aplikaci jako daemona.

3.5.5 Interakce s Vert.x

Díky modulu CrasHub Shell⁷ se lze protokolem SSH⁸ přihlásit přímo do Vert.x. Modul pak nabízí možnost interakce s jednotlivými komponentami samotného Vert.x. Lze například posílat zprávy přes Event Bus nebo nasazovat nové moduly za běhu celé aplikace.

Pro nasazení modulu stačí přidání klíče *shell* do konfigurace server 2. Po nasazení aplikace začne tento server poslouchat na portu 2000.

Kód 3.13: Konfigurace modulu CrasHub Shell

```
{
  "name": "MindMap editor server 2 databázový modul, obrazkový
    exporter",
  "shell": {
    "crash.auth": "simple",
    "crash.auth.simple.username": "admin",
    "crash.auth.simple.password": "heslo",
    "crash.ssh.port": 2000
  }
}
```

⁷<https://github.com/crashub/mod-shell>

⁸Secure Shell

Samotný modul nabízí nepřehledné množství možností jak spravovat aplikace nebo ji naživo škálovat. Lze také jednoduše přidat vlastní příkazy a rozšířit tak možnosti tohoto modulu. Na obrázku 3.7 je hlavní přehledová stránka na které lze vidět činnost, vytížení a status všech vláken v celém clusteru. Dostupné jsou také informace o velikosti zásobníku, paměti či verze Javy.

Škálování je nedílnou součástí životního cyklu aplikace. Ne zřídka dojde aplikace do situace, kdy začne být pomalá či často padat pod velkým nápořem klientů. Následující kapitola rozebírá možnosti škálování Vert.x aplikací.

Samotné vertikální škálování lze efektivně řešit až na aplikační úrovni. Jak již bylo zmíněno v kapitole 2.2.2 voláním `Runtime.getRuntime().availableProcessors()` lze získat počet procesorových jader a s tím dále pracovat. Upravením předchozích příkazů však docílíme shodného výsledku.

38

3.6.2 Horizontální

Dle návrhu architektury na obrázku 3.2 bude aplikace nasazena na dva servery. První bude naslouchat na port 80 a poběží zde Webový server(HTTP Server). Tento server má vnitřní IP adresu *10.10.10.161*. Druhé rozhraní má připojené do internetu. Na druhém serveru je část aplikace, která komunikuje s databází a modul pro vzdálenou interakci s Vert.x. Jeho IP adresa je *10.10.10.162*. Pro propojení obou instancí je potřeba upravit spouštěcí příkaz v konfiguraci Supervisoru.

Kód 3.14: Spuštění clusteru na Serveru 1

```
command=vertx runmod io.majklk~mindmapeditor~0.0.1 -conf
/srv/mindmap/conf/webserver.json -cluster -cluster-host
10.10.10.161
```

Kód 3.15: Spuštění clusteru na Serveru 2

```
command=vertx runmod io.majklk~mindmapeditor~0.0.1 -conf
/srv/mindmap/conf/dbserver.json -cluster -cluster-host
10.10.10.162
```

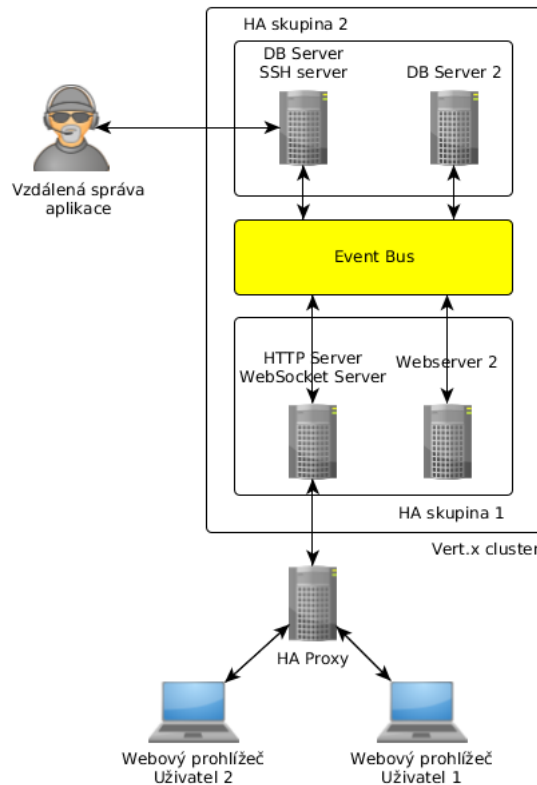
3.6.3 Ladění výkonnosti

Hlavní sada vláken, tedy event loopů je ve výchozím nastavení na hodnotě odpovídající volání funkce *Runtime.getRuntime().availableProcessors()*. Tuto hodnotu lze změnit nastavením systémové proměnné *vertx.pool.eventloop.size*. Nastavením *vertx.pool.worker.size* pak lze změnit velikost poolu pro dlouhotrvající operace, která je ve výchozím nastavení na hodnotě 20.

3.7 Vysoká dostupnost

Pro zajištění vysoké dostupnosti klíčových prvků aplikace, je potřeba upravit architekturu clusteru. Před webový server je postaven load balancer⁹ v tomto případě HA proxy, která při úpadku jednoho z webových serverů přesměruje komunikaci na server druhý. Vert.x cluster je pak rozdělený na dvě HA skupiny(obr.3.8), které se liší svým zaměřením. První dva servery slouží jako webové a jsou napojeny na HA proxy. Další dva pak slouží pro komunikaci s databází, která na nich, přímo běžet nemusí. V této HA skupině je pak dále modul pro interakci s Vert.x. Díky specifikování HA skupiny nikdy nedojde k nasazení modulu na webovém serveru a tedy otevření SSH na portu 2000.

⁹ služba zajišťující vyrovnavání zatížení



Obrázek 3.8: Ideální architektura nasazení aplikace

Kód 3.16: Vysoká dostupnost na databázovém serveru 2

```
command=vertx runmod io.majklk~mindmapeditor~0.0.1 -conf
/srv/mindmap/conf/dbserver.json -cluster-host 10.10.10.162 -ha
-hagroup skupina-1
```

Když je specifikován parametr *-ha* lze automaticky vypustit parametr *-cluster*.

3.8 Integrace do stávající Java aplikace

Pokud je to jakkoliv jen možné je vhodné se integraci vyhnout. Pokud však nejde jinak, jsou zde dvě varianty jak integrovat platformu.

1. PlatformManager
2. Pomocí jar souboru

Kód 3.17: Integrace do stávající Java aplikace

```
//díky platform managerovi lze provádět stejné úkony jako v
//příkazové řádce
PlatformManager pm = null;
pm = PlatformLocator.factory.createPlatformManager();

JsonObject conf = new JsonObject().putString("foo", "wibble");

pm.deployModule("com.mycompany~my-module~1.0", conf, 10, new
    AsyncResultHandler<String>() {
        public void handle(AsyncResult<String> asyncResult) {
            if (asyncResult.succeeded()) {
                System.out.println("Deployment ID is " +
                    asyncResult.result());
            } else {
                asyncResult.cause().printStackTrace();
            }
        }
    }
});
```

V takovém případě bude platforma hledat *cluster.xml* a *repos.txt* v proměnné classpath.

4 Závěr

Vert.x je vysoce modifikovatelný webový framework založený na komunikaci v reálném čase napříč všemi částmi aplikace. Vysoká modularita a otevřenost platformy Vert.x přináší značné výhody. Již od počátku si kladl za cíl zjednodušit dosavadní možnosti vývoje a představit tak alternativu ke standardním nástrojům vývoje webových aplikací. Je to právě jednoduchost, univerzálnost a komplexnost řešení této platformy, které zláká nejednoho programátora aby minimálně zaexperimentoval s tímto nástrojem. V současné době se velmi progresivně rozšiřuje celý ekosystém okolo Vert.x novými nástroji a možnostmi.

Práce představila unikátní filosofii a principy frameworku Vert.x a snažila se do jisté míry provést začínajícího uživatele hlavními aspekty vývoje s poukázáním na další související problematiku. Přitom se maximálně snažila omezit pouze na oblast vývoje webových aplikací, přestože někdy nebylo možné opomenout elementární souvislosti s vývojem distribuovaných webových aplikací. V práci byla představena platforma Vert.x jako nástroj pro distribuované webové aplikace. Byla popsána unikátní filozofie a terminologie této platformy.

V praktické části se podařilo vytvořit webovou aplikaci, která splňuje všechny aspekty moderní webové aplikace. Především pak komunikace v reálném čase bez náročných implementací či použití mnoha služeb a nástrojů. V aplikaci je možné jednoduchým a intuitivním způsobem přidávat, přejmenovávat a odebírat její jednotlivé body. Pokud má stejnou myšlenkovou mapu otevřeno více lidí, okamžitě vidí všechny změny, ostatních klientů. Aplikace používá volně šiřitelný software, který je ve většině případů špičkové úrovně. Možnosti pro vylepšení aplikace jsou jak na straně vizuální tak na straně funkcionální. Bylo by vhodné rozšířit aplikaci o možnost přihlášení a správy pouze svých myšlenkových či případné sdílení jednotlivých map s ostatními uživateli.

Z práce vyplývá, že se platforma Vert.x hodí pro vývoj webových aplikací, výhradně pak za použití s dalšími nástroji usnadňující vývoj MVC případně MVVM aplikací, například Spring frameworkem.

4.1 Možnosti dalšího výzkumu

Tak rozsáhlé téma jako jsou distribuované webové aplikace rozhodně nelze podrobně popsat v rámci jedné bakalářské práce. Na tuto práci proto mohou navazovat kolegové z fakulty či jiných vysokých škol. V závěru pro ně přináším dva zajímavá témata, na které již v této práci nezbyl prostor a rozhodně si zaslouží podrobnější analýzu.

4.1.1 Distribuované výpočty

4.1.2 Srovnání

Literatura

- [1] Phipps, Simon *Who controls Vert.x: Red Hat, VMware, or neither?* [online]. [cit. 2014-02-16]. Dostupný z WWW: <http://www.infoworld.com/d/open-source-software/who-controls-vertx-red-hat-vmware-or-neither-210549>
- [2] vertxNodejs *An Interview with Tim Fox – Vert.x and why it's better than NodeJS!* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.jclarity.com/topics/interviews/tim-fox>
- [3] Maurer, Norman *Netty in Action* [online]. [cit. 2014-03-20]. Dostupný z WWW: http://www.manning.com/maurer/netty_meap_ch1.pdf
- [4] Kamali, Masoud *The Winners of the JAX Innovation Awards 2014* [online]. [cit. 2014-03-20]. Dostupný z WWW: <http://jax.de/awards2014/>
- [5] Gardoh, Ed *Parallel Processing and Multi-Core Utilization with Java* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://embarcadero.net/2011/01/23/parallel-processing-and-multi-core-utilization-with-java/>
- [6] Merta, Zdeněk *Vert.x jOpenSpace 2013* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://jopenspace.cz/2013/presentations/zdenek-merta-vert.x.pdf>
- [7] Ponge, Julien *Fork and Join: Java Can Excel at Painless Parallel Programming Too!* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.oracle.com/technetwork/articles/java/fork-join-422606.html>
- [8] *Package java.util.concurrent Description* [online]. [cit. 2014-03-22]. Dostupný z WWW: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html#package_description
- [9] Sun Song, Ki *Understanding Vert.x Architecture - Part II* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.cubrid.org/blog/dev-platform/introduction-to-in-memory-data-grid-main-features/>

-
- [10] Jaehong, Kim *Introduction to In-Memory Data Grid: Main Features* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.cubrid.org/blog/dev-platform/understanding-vertx-architecture-part-2/>
- [11] Pitner, Tomáš *Programování v jazyce Java* [online]. [cit. 2014-04-10]. Dostupný z WWW: <http://www.fi.muni.cz/~tomp/slides/pbl62/printable.html>
- [12] Lažanský, J. *Procesy a vlákna* [online]. [cit. 2014-04-15]. Dostupný z WWW: <http://labe.felk.cvut.cz/vyuka/A4B33OSS/Tema-03-ProcesyVlakna.pdf>
- [13] Fox, Tim *Event loops* [online]. [cit. 2014-04-15]. Dostupný z WWW: <http://vertx.io/manual.html#event-loops>
- [14] Kosek, Jiří *Session proměnné* [online]. [cit. 2014-04-15]. Dostupný z WWW: <http://www.kosek.cz/clanky/php4/session.html>
- [15] Janssen, Cory *Message Queue* [online]. [cit. 2014-04-22]. Dostupný z WWW: <http://www.techopedia.com/definition/25971/message-queue>
- [16] Froemke, Dina *Framework Benchmarks Round 8* [online]. [cit. 2014-04-22]. Dostupný z WWW: <http://www.techempower.com/blog/2013/12/17/framework-benchmarks-round-8/>
- [17] Fox, Tim *Vert.x vs node.js simple HTTP benchmarks* [online]. [cit. 2014-06-22]. Dostupný z WWW: <http://vertxproject.wordpress.com/2012/05/09/vert-x-vs-node-js-simple-http-benchmarks/>
- [18] Osuszek, Lukasz *Distributed Architecture of Enterprise Information Systems* [online]. [cit. 2014-07-31]. Dostupný z WWW: <http://www.soainstitute.org/resources/articles/distributed-architecture-enterprise-information-systems>
- [19] Halterman, Jordan *Vertigo* [online]. [cit. 2014-07-31]. Dostupný z WWW: <https://github.com/kuujo/vertigo>
- [20] Bostock, Mike *Collapsible Tree* [online]. [cit. 2014-07-31]. Dostupný z WWW: <http://bl.ocks.org/mbostock/4339083>
- [21] Canonical Ltd. *Ubuntu Server Edition* [online]. [cit. 2014-08-01]. Dostupný z WWW: <http://www.ubuntu.com/server>

- [22] Williams, Alex *MongoDB Raises 150M For NoSQL Database Technology With Salesforce Joining As Investor* [online]. [cit. 2014-08-02]. Dostupný z WWW <http://techcrunch.com/2013/10/04/mongodb-raises-150m-for-nosql-database-technology-with-salesforce-joining/>
- [23] Rauch, Guillermo *WebSocket emulation* [online]. [cit. 2014-08-02]. Dostupný z WWW <https://github.com/sockjs>
- [24] Malý, Martin *Web sockets* [online]. [cit. 2014-08-02]. Dostupný z WWW <http://www.zdrojak.cz/clanky/web-sockets/>
- [25] Oracle *Naming a Package* [online]. [cit. 2014-08-02]. Dostupný z WWW <http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html>

Přílohy

Seznam obrázků

2.1	Architektura Vert.x převzato a upraveno z [10]	4
2.2	Blokující přístup <i>převzato a upraveno z [3]</i>	7
2.3	Neblokující přístup pomocí Netty.io <i>převzato a upraveno z [3]</i>	8
2.4	Vert.x instance	9
2.5	Příklad vertikálního škálování <i>vertx run HelloWorld -instances 4</i>	11
2.6	Event Bus distribuovaný mezi dva servery	12
2.7	Horizontální a vertikální škálování	16
2.8	Clustering mezi třemi Vert.x instancemi	18
2.9	Výsledky prvního testu <i>Tim Fox</i> [17]	20
2.10	Výsledek druhého testu <i>Tim Fox</i> [17]	22
3.1	Případy užití	24
3.2	Architektura nasazené aplikace	24
3.3	Komunikace v reálném čase	30
3.4	Webová aplikace	34
3.5	Webová aplikace - otevření myšlenkové mapy	35
3.6	Webová aplikace - akce - editace	35
3.7	Modul CrasHub Shell	38
3.8	Ideální architektura nasazení aplikace	40

Seznam tabulek

2.1	Srovnání odezvy, převzato a upraveno z [16]	21
2.2	Srovnání vlastností s Node.js	21

List of Listings

3.1	Spuštění modulu z příkazové řádky	25
3.2	Konfigurace serveru 1	25
3.3	Zaregistrování obslužné rutiny v jazyce JavaScript	26
3.4	Zaregistrování obslužné rutiny v jazyce Java	27
3.5	Publikování zprávy v jazyce JavaScript	27
3.6	Uložení myšlenkové mapy do databáze	27
3.7	Konfigurace serveru 2	28
3.8	Připojení Event busu z prohlížeče a inicializace editoru	31
3.9	D3.js nicializace dat	32
3.10	Spuštění modulu v jazyce Java	33
3.11	Verticle v jazyce Java	33
3.12	Konfigurace produkční služby	37
3.13	Konfigurace modulu CrasHub Shell	37
3.14	Spuštění clusteru na Serveru 1	39
3.15	Spuštění clusteru na Serveru 2	39
3.16	Vysoká dostupnost na databázovém serveru 2	40
3.17	Integrace do stávající Java aplikace	40