

Děkuji doc. Ing. Filip Malému, Ph.D. za odborné vedení práce, poskytování rad.

Tato stránka je tzv. protititul a je graficky součástí titulní stránky. Nechte ji prázdnou, nebo na ni umístěte vhodnou fotografii či ilustraci.

UNIVERZITA HRADEC KRÁLOVÉ
FAKULTA INFORMATIKY A MANAGEMENTU
KATEDRA INFORMATIKY A KVANTITATIVNÍCH METOD
2014-06-29



BAKALÁŘSKÁ PRÁCE

Vert.x jako platforma pro webové aplikace

Autor: Michael Kutý

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

Hradec Králové, 2014

Anotace Bakalářská práce se zaměřuje na problematiku vývoje distribuovaných webových aplikací. Teoretická část práce popisuje architekturu platformy Vert.x a problémy, které tato platforma řeší. V praktické části bude implementovaná malá jednostránková kolaborativní aplikace jejíž jednotlivé části budou roz distribuované na více instancí aby byla zajištěna vysoká dostupnost. Aplikace se nasadí do dvou referenčních instalací. První do prostředí VirtualBox a druhá v prostředí laboratoře CEPSES při UHK.

Annotation English content

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

V Kroměříži dne 25. července 2014

Michael Kutý

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 Multi-reactor pattern	6
2.2.3 Terminologie	7
2.2.4 Event Bus	10
2.3 API	12
2.3.1 Základní API	13
2.3.2 Kontainer API	13
2.3.3 Polyglot	13
2.4 Clustering	14
2.4.1 Vysoká dostupnost	14
2.5 Porovnání s Node.js	16
2.5.1 Výkon	16
2.5.2 Vlastnosti	18
2.5.3 Závěr srovnání	18
3 Praktická část	20
3.1 Návrh	20
3.1.1 Cíle aplikace	21
3.1.2 Architektura	21
3.2 Řídící verticle	21
3.3 Integrace s databází MongoDB	21
3.4 Komunikace v reálném čase	21
3.4.1 Akce	22
3.4.2 Události	22
3.5 Polyglot vývoj a moduly	22
3.6 Nasazení	25
3.6.1 Server	25
3.6.2 Java	25
3.6.3 Vert.x	25

3.6.4	MongoDB	25
3.7	Škálování a vysoká dostupnost	25
3.7.1	Počet Verticlů	25
3.7.2	Interakce s Vert.x	26
3.7.3	Vert.x v clusteru	26
3.7.4	Vysoká dostupnost	27
4	Závěr	29
	Literatura	31
	Přílohy	I

1 Úvod

V současné době existuje nespočet frameworků¹ pro vývoj webových aplikací ve spoustě programovacích jazycích. Vývojář tak nemá vůbec lehké, vybrat ten správný nástroj, který by mu zaručil, že se jeho aplikace dostane na hranici možností, které mu daný nástroj poskytuje.

Většina webových aplikací ovšem dříve nebo později narazí na problematiku škálování, kdy je třeba rozložit aplikaci na více serverů ať už pro zajištění vysoké dostupnosti nebo co nejnižší odezvy. Dnes také není nic neobvyklého, že aplikaci najednou začnou navštěvovat tisíce klientů za minutu a rázem se tak může stát, že z jinak rychlé aplikace se stane 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 motivovali k hlubšímu studiu platformy Vert.x. V hlavní části práce následuje návrh a vlastní implementace 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 jednostránkových aplikací dále jen SPA. Vytvoření jednoduchého webového editoru myšlenkových map dále jen mindmap. Na této jednoduché aplikaci bude demonstrován proces vývoje webové aplikace pod platformou Vert.x. Při vývoji klientské části bude použit návrhový vzor MVVC.

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. A to jak událostmi řízenou architekturou, kterou platforma převzala z dnes již dobře známého frameworku Node.js. Tak především polyglot programování s jednoduchým konkurenčním modelem a možnost sdílet data mezi jednotlivými vlákny bez nutnosti zámků.

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

Cílem teoretické části je tedy popsat jednotlivé části platformy a jejich účel či problém, který řeší. V závěru teoretické části bude platforma srovnána s již zmíněným nástrojem Node.js² to v několika důležitých aspektech rychlosti, která je v dnešním světě neustálého růstu počtu zařízení, to co trápí webové aplikace s desítkami tisíc dlouho trvajících připojení.

V praktické části bude vytvořen editor pro jednoduchou správu a tvorbu mindmap. Tyto mindmapy bude moct upravovat více uživatelů najednou v reálném čase. Budou popsány a vysvětleny jednotlivé kroky vývoje až po úplné nasazení webové aplikace na jednotlivé servery, kde bude prověřena funkčnost distribuovaného provozu aplikace. Pro nasazení aplikace na více serverů bude použit nástroj konfiguračního managementu Salt Stack.

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. Pro realizaci webové aplikace budou použity pokročilé techniky, které učiní aplikaci ještě více znovupoužitelnou a škálovatelnou. Tyto techniky budou čtenáři vysvětleny podrobným způsobem s použitím ukázek. 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:

- Java Development Kit 7: soubor základních nástrojů a knihoven pro běh a vývoj Java aplikací.
- Ubuntu 12.04: operační systém vhodný pro běh Vert.x aplikací
- Vert.x 2.1M3+: platforma pro vývoj webových aplikací
- MongoDB: dokumentové orientovaná NoSQL³ databáze
- D3.js: framework pro práci s grafy

²Serverový framework, postavený na modelu událostmi řízeného programování

³databázový 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 real-time kolaborativní aplikace, které drasticky změnilly potřeby programátorů, na jednotlivé nástroje. Programátor tak má možnost zvolit si z velké řádky nástrojů mezi než patří například Node.js, Akka či ruby EventMachine. Problémem těchto jinak časem a komunitou prověřených platforem může být fakt, že jsou úzce spjaté s konkrétním programovacím jazykem či velmi náročná integrace do již stávající aplikace.

Vert.x je projekt vycházející z Node.js, který jako první framework, pokořil v roce 2010 C10K¹ problém. Platforma Vert.x má velice podobné API² jako Node.js. Obě platformy poskytují kompletně asynchronní API. Jak již název napovídá Node.js je napsán v JavaScriptu, zatím co Vert.x je implementován v Javě. Vert.x ale není pouhá reimplementace Node.js do jazyka Java. Platforma má svou vlastní unikátní filozofii, která je diametrálně odlišná od Node.js.

2.1 Historie

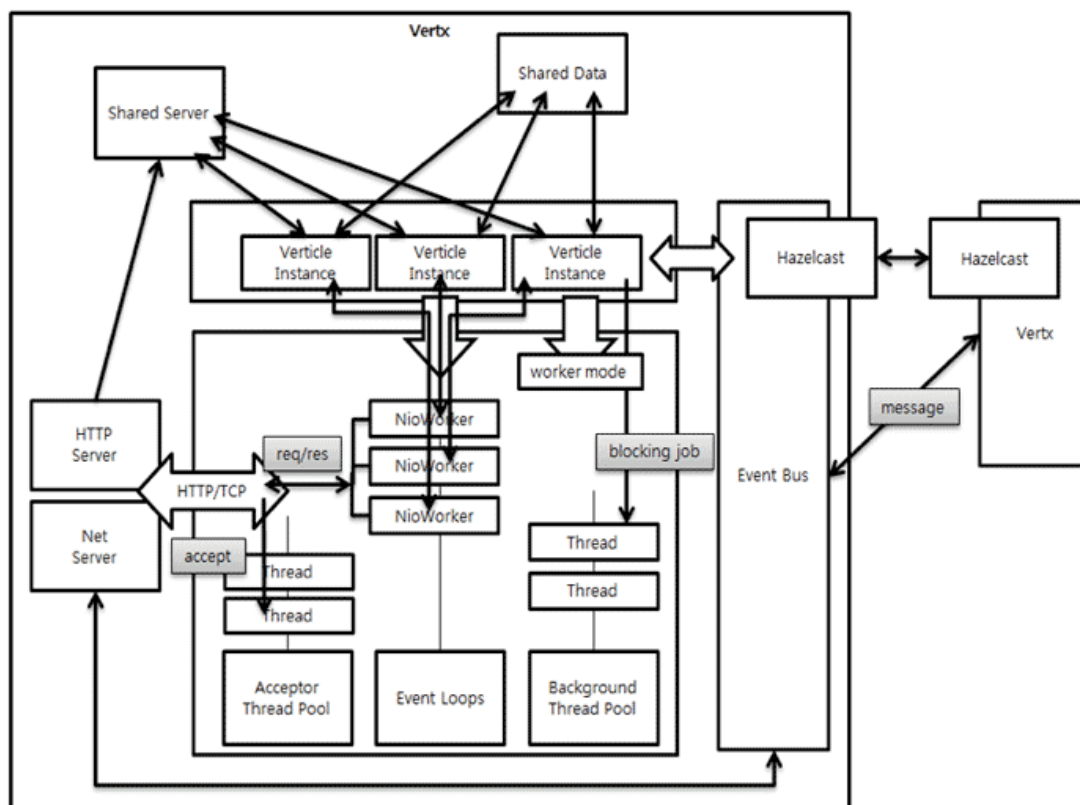
Začátek vývoje projektu Vert.x je datován do roku 2011. Tedy rok poté co spatřil světlo světa framework Node.js a za pouhý rok si vydobyl své místo u komunity, která si jej velmi oblíbila. Pravděpodobně největší motivací pro vývoj nové platformy podobné Node.js byla právě oblíbenost Node.js.

Hlavním autorem platformy byl a 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 budoucí zdravý rozvoj platformy bude přesunutí celé

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

²Application Programming Interface

Obrázek 2.1: Architektura Vert.x *Jaehong Kim*

platformy pod nadací Eclipse Foundation, k čemuž došlo na konci roku 2013. V dnešní době se platforma těší velkému vývoji, který čítá desítky pravidelných přispěvatelů mezi něž patří mimo Tima například také Norman Maurer, který patří mezi přední inženýry vyvíjející framework Netty.io, který zodpovídá za integraci Netty frameworku do Vert.x platformy.

Na tomto místě by bylo vhodné uvést, že platforma Vert.x letos vyhrála prestižní cenu "Most Innovative Java Technology" v soutěži JAX Innovation awards[2].

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 rozebrána v následujících kapitolách.

2.2.1 Jádru

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 činnost.

Lze jej následně rozšířit o novou funkčnost dokompilováním balíčků, které lze naléznout v oficiálním repositáři. Pravděpodobnou inspirací byl již zmíněný Node.js respektive NPM³ u kterého se takováto forma vývoje velice oblíbila. Od doby vzniku této platformy vzniklo nespočet rozšíření, které udělaly z Node.js silný nástroj pro rychlý vývoj webových aplikací. Klíčové jsou aspekty jako událostmi řízené programování a neblokující asynchronní model.

Asynchronní model

Událostmi řízené programování je podle Tomáše Pitnera[9] základním principem tvorby aplikací s GUI(Graphical user interface). Netýká se však pouze GUI, je to obecnější pojem označující typ asynchronního programování, kdy je: tok programu řízen událostmi na které navěšuje tzv. event handlers⁴.

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 Vlákno⁵ je 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.

³Node package manager

⁴obslužná rutina události

⁵Označuje v informatice odlehčený proces, pomocí něhož se snižuje režie operačního systému při změně kontextu, které je nutné pro zajištění multitaskingu

Existují dva druhy asynchronního modelu (multitasking): multiprocesorový: o běh, tvorbu a režii vláken se stará operační systém multivláknový: o běh, tvorbu a režii vláken se stará aplikace a předává je operačnímu systému Podle Lažanského[10] je sdílení paměti důsledkem nižší režie při přepínání (přepnutí vláken je výrazně rychlejší), obdobně i vytváření a rušení vlákna a samozřejmě i úspora paměti. Jak již bylo zmíněno jádro Vert.x je implementováno v jazyce Java a pro Vert.x je tedy důležité, jak moc je dobrá implementace paralelního modelu v jazyce JAVA. Zde se dostáváme k jedinému požadavku pro běh Vert.x instancí a to 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`

CyclicBarrier⁶ z balíčku `java.util.concurrent`

CountDownLatch z balíčku `java.util.concurrent`

File z balíčku `java.nio`

Vylepšený ClassLoader⁷ lepší odolnost vůči deadlockům⁸

Více o `java.concurrent`[6]

Ed Gardoh v roce 2011 ve svém jednoduchém testu[3] prověřil práci s paralelizací úkonů. Z jeho testů vyplývá, že Java 1.7 je až o 40% rychlejší při práci s vlákny díky nové metodě `Fork/Join`[5].

2.2.2 Multi-reactor pattern

Základ jádra je postaven na tzv. Multi-reactor pattern[11], který vychází z Reactor patternu[4], ten lze charakterizovat několika body:

- aplikace je řízena událostmi
- na události se registrují handlery
- vlákno zpracovává události a spouští registrované handlery
- toto vlákno nesmí být blokováno⁹

⁸ 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.

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

Multi-reactor pattern[11] 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. Hlavnímu vláknu, se ve Vert.x komunitě říká *Event Loop*. V komunitách Nginx nebo Node.js se ovšem setkáme spíše s pojmem *Run Loop*. Nevýhoda tohoto modelu je, že nikdy nesmí dojít k blokování hlavního vlákna a také fakt, že platforma Node.js poskytovala jenom jedno vlákno, které šlo škálovat na jednotlivé procesory. Jak je vidět z obrázku 2.2 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 3.7. Na obrázku 2.3 na straně 9 pak lze vidět situaci čtyř hlavních vláken na čtyři procesorové jádra.

Příklady blokujících volání:

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

Hybridní model vláken

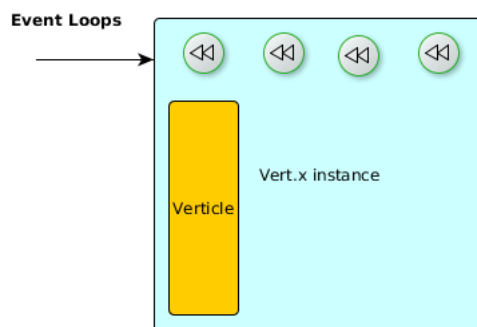
Platforma Vert.x přišla s inovací v oblasti hlavních vláken a to takovou, že k hlavním *Event loops* přidala další sadu vláken *Background thread pool*, které jsou vyčleněny z hlavní architektury a poskytují samostatnou kapitolu pro škálování aplikace. To lze ostatně vidět na obrázku 2.1 na straně 4. 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.

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 v jazyce Java pak jako 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. Díky tomu, že každý verticle běží ve vlastním vlákně odpadá nutnost zámků nad proměnnými a nejrůznější synchronizace vláken či deadlocky se tak stávají minulostí. Vzhledem k



Obrázek 2.2: Vert.x instance

tomu, že každý verticle má svůj vlastní classloader nemůže tak sdílet statické metody ani hodnoty proměnných s ostatními. Sdílet data je tak možné pouze dvěma způsoby.

- pomocí Message Queue[13] dále jen MQ
- SharedData object a SharedSet *vertx.sharedData()*

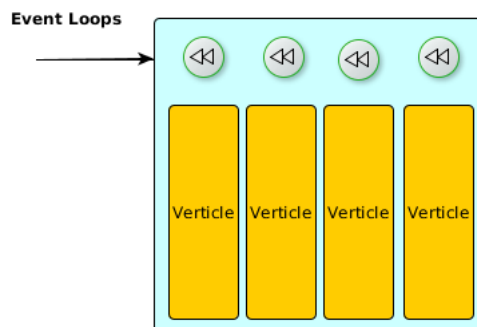
Objekty v SharedData musí být immutable¹⁰. V dnešní době je řada MQ frameworků přes které lze vést komunikaci u platformy Vert.x však není potřeba externí služba, protože má vlastní Event Bus o kterém pojednává kapitola 2.2.4. Na obrázku 2.2 je vidět jeden verticle v kontextu jedné Vert.x instance. Následuje sumarizace vlastností verticle.

- nejmenší spustitelná jednotka
- 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. Tím dojde k vyčlenění verticle z asociace na hlavní vlákna a takovému vlákně pak bude přiděleno vlákno z Background thread poolu. Uvnitř takového to verticle lze pak vykonávat blokující volání bez blokování

¹⁰jakmile jednou takovýto objekt vznikne nejde dále měnit jeho proměnné



Obrázek 2.3: Vert.x instance `vertx run HelloWorld -instances 4`

celé aplikace. To se v praxi ukázalo jako velice užitečná věc. Bohužel tímto ztrácíme efektivní možnost škálování pro velký počet konkurenčních vláken.

Vert.x instance

Každý verticle běží uvnitř Vert.x instance 2.2 na předchozí straně a každá instance běží ve vlastní JVM instanci. V jedné Vert.x instanci může najednou běžet nespočet Verticlů. Všechny verticle mohou běžet souběžně na jednom serveru. Na jedno serveru může současně běžet mnoho Vert.x instancí v případě clusterování i na více serverech. verticle spolu pak komunikují pomocí distribuovaného EventBusu.

Moduly

Moduly poskytují možnost zapouzdření a znovupoužitelnost funkcionality. V praxi se mohou moduly skládat z více modulů či verticlů ve více programovacích jazycích. Moduly mohou být uloženy v centrálním repozitáři¹¹ nebo může být využit jakýkoliv jiný repozitář. Repozitáře v kterých hledá Vert.x při startu instance dostupné moduly lze definovat v hlavní konfiguraci Vert.x. Každý modul musí mít svůj deskriptor ve formátu JSON¹². Jak může vypadat deskriptor je více popsáno v kapitole 3.5.

Výhody plynoucí z použití modulů:

- classpath¹³ je zapouzdřený a díky tomu lze moduly používat mnohem snáze
- všechny závislosti jsou zapouzdřeny v jediném souboru ve formátu ZIP
- moduly mohou být umístěny v repozitářích

¹¹<http://modulereg.vertx.io/>

¹²je odlehčený formát pro výměnu dat

¹³říká JVM, kde má hledat třídy a balíčky

- 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 použitím metody *includes*

2.2.4 Event Bus

Nervový systém celého Vert.x, jehož název lze volně přeložit jako sběrnice událostí. Cílem EventBusu je zpožděkování komunikace mezi jednotlivými komponentami a vlákny 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. Obrovskou výhodou oproti externí MQ je fakt, že lze takovouto komunikaci snadno přemostit ke klientovi na straně webového prohlížeče což je detailněji popsáno v kapitole 3.4.

Základní typy komunikace:

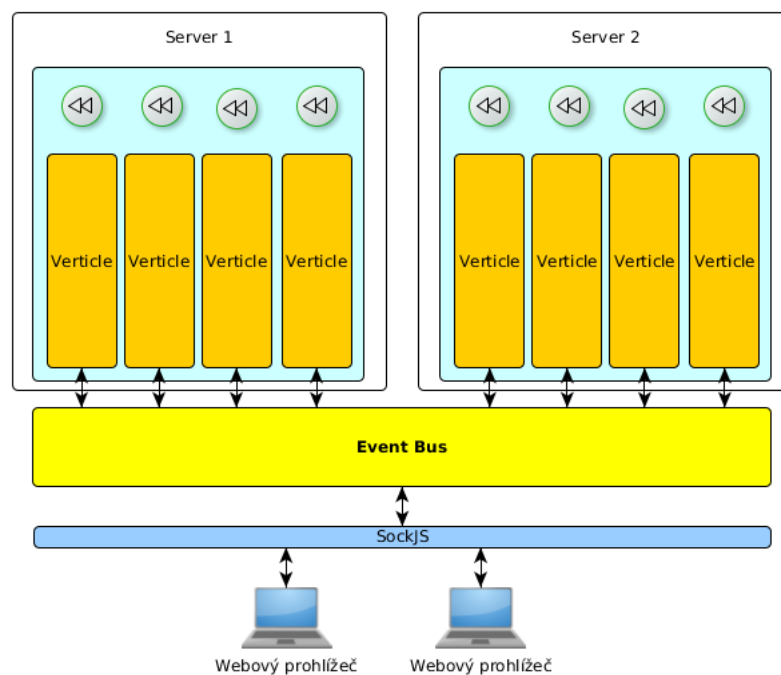
- Point to Point
- Publish/Subscribe

typy zpráv:

- String
- primitivní typy (int, long, short, float double, ..)
- 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`¹⁴ rozšíří EventBus o možnost používat mnohem komplexnější typy zpráv jejichž výčet se nachází níže.

¹⁴<https://github.com/pmlopes/mod-bson-io>



Obrázek 2.4: Event Bus distribuovaný mezi dva servery

- `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 2,6MB kódu v jazyce Java. Hlavní výhody In-memory data grid[7] 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 lze využít v několika rolích:

- NoSQL databáze v paměti
- Cache¹⁵
- Data grid
- Zasílání zpráv
- Aplikační škálování
- Clustrování aplikací

Hazelcast je tedy typ distribuovaného úložiště, které běží jako vestavěný systém a lze díky němu distribuovat celou aplikaci do více geografických lokací nebo zasílat zprávy mezi jednotlivými komponentami. Vert.x API využívá Hazelcast API a odstiňuje tak programátora od poměrně nízko úrovně API Hazelcastu. Když je Vert.x spuštěn, Hazelcast je spuštěn v módu vestavěného systému. Odpadá tak režie další služby. Jako nejčastější příklad užití samotného Hazelcastu bývá uváděno ukládání uživatelské session[12]. 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.

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 kdokoliv 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*.

¹⁵specializovaný typ paměti pro krátkodobé ukládání

2.3.1 Základní API

Základní API, které Vert.x poskytuje programátorovi je poněkud strohé a obdobné jako u frameworku Node.js. Platforma tak poskytuje stabilní základ, který se v praxi neobejde bez modulů o kterých pojednává kapitola 2.2.3.

- TCP/SSL server/klient
- HTTP/HTTPS server/klient
- Websockets server/klient, SockJS
- Distribuovaný Event Bus
- Časovače
- Práce s buffery
- Přístup k souborovému systému
- Přístup ke konfiguraci

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 CLI¹⁶.

- 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 apod. Tento fakt

¹⁶Command Line Interface

hodně napomáhá celé platformě nalákat nové programátory, protože ne každý na světě umí programovací jazyk Java. Výčet podporovaných jazyků ve verzi 2.0. Do verze 3.0 se pak chystá automatické generování API pro každý jazyk.

- Java
- Javascript, CoffeeScript
- Ruby
- Python
- Groovy
- PHP
- Clojure

2.4 Clustering

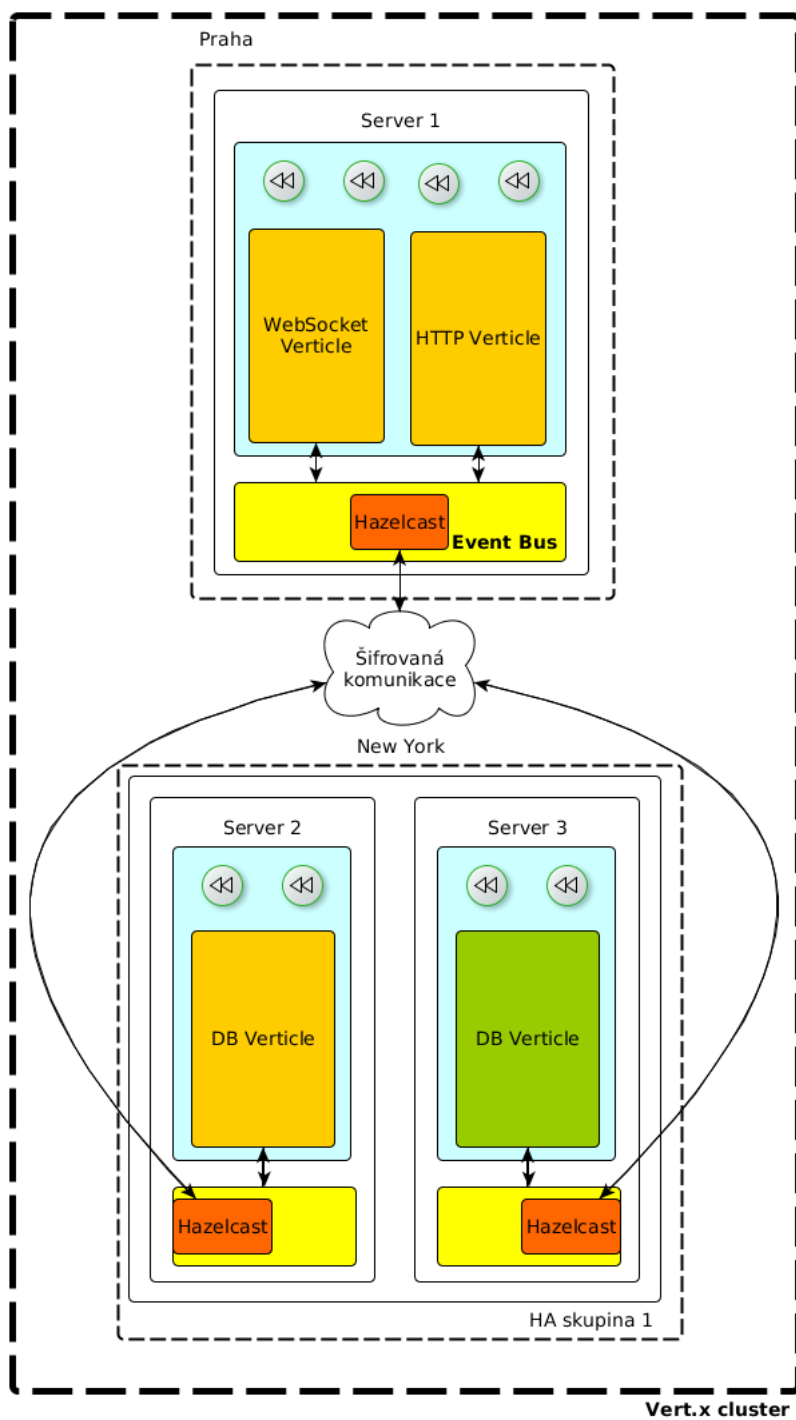
Díky integraci Hazelcastu získala platforma Vert.x řadu zajímavých vlastností mezi které patří také možnost vertikálního škálování neboli clusteringu. To v praxi znamená, že můžeme aplikace jednoduše škálovat přes více serverů bez nutnosti běhu dalších služeb a režijních nákladů. Samotná konfigurace clusteru není pak nic složitějšího a odehrává se v souboru *conf/cluster.xml* a spočívá v nastavení členů clusteru nebo specifikování multicastové¹⁷ adresy a portu na které bude Hazelcast po startu vyhledávat členy clusteru. Velkou výhodou je pak možnost šifrované komunikace, díky čemuž odpadá nutnost použití nejrůznějších služeb zajišťující šifrování komunikace na síťové vrstvě v případě nasazení na veřejné síti nebo ve více geografických lokacích. Pro spuštění aplikace v režimu cluster ji stačí spustit s parametrem *-cluster*. Více o konfiguraci clusteringu v kapitole 3.7.3.

2.4.1 Vysoká dostupnost

Samostatnou kapitolou v oblasti clusteringu je HA¹⁸ česky tedy vysoká dostupnost. Díky Hazelcastu ji lze řešit již na aplikační úrovni, a není potřeba dalších služeb, které řeší vysokou dostupnost.

¹⁷logický identifikátor skupiny síťových hostů

¹⁸HA - High Availability



Obrázek 2.5: Clustering mezi třemi Vert.x instancemi

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 2.5 v případě pádu Serveru 2, tedy části aplikace, která komunikuje s databází dojde automaticky k novému nasazení této části do nové instance.

Skupiny HA

V případě spuštění modulů v režimu HA lze pak specifikovat logické skupiny. Díky tomu lze určit, kde se mají moduly v případě pádu nasadit. Z toho logicky vyplývá, že moduly se nasadí jen na instancích se stejnou HA skupinou.

Kvorum

Při spuštění Vert.x instance lze specifikovat kvorum¹⁹. 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.

2.5 Porovnání s Node.js

V následující kapitole bude porovnána platforma Vert.x s již zmíněnou platformou Node.js. Výkonnostní test[15] 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[14].

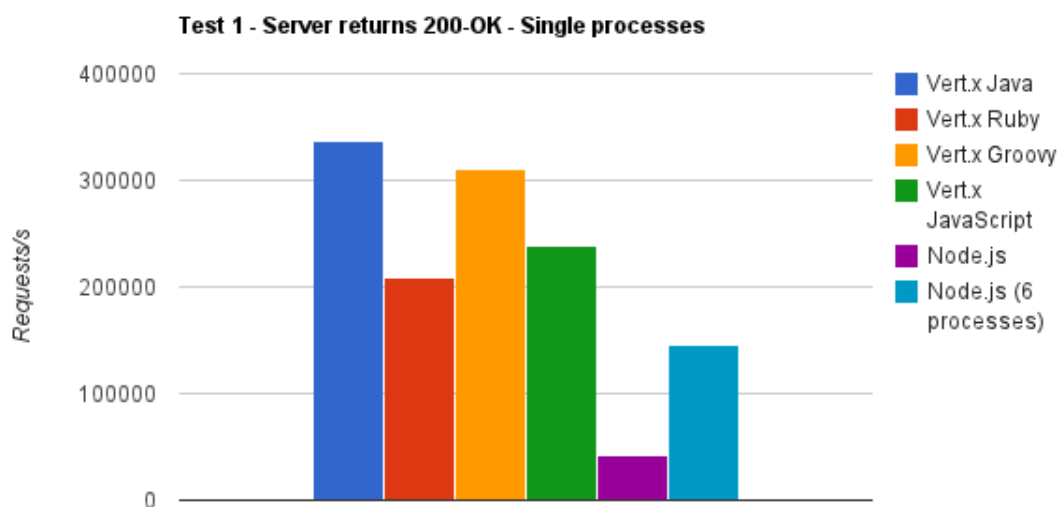
2.5.1 Výkon

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

¹⁹ minimální počet serverů pro zajištění vysoké dostupnosti



Obrázek 2.6: Výsledky druhého testu *Tim Fox*

variantách. Samostatná a šest procesů v jednom clusteru. V legendě grafů je to odlišeno příponou *cl*.

1. Triviální dotazování serveru a návrat statusu 200²⁰
2. Dotaz na statický soubor o velikosti 72 bytů

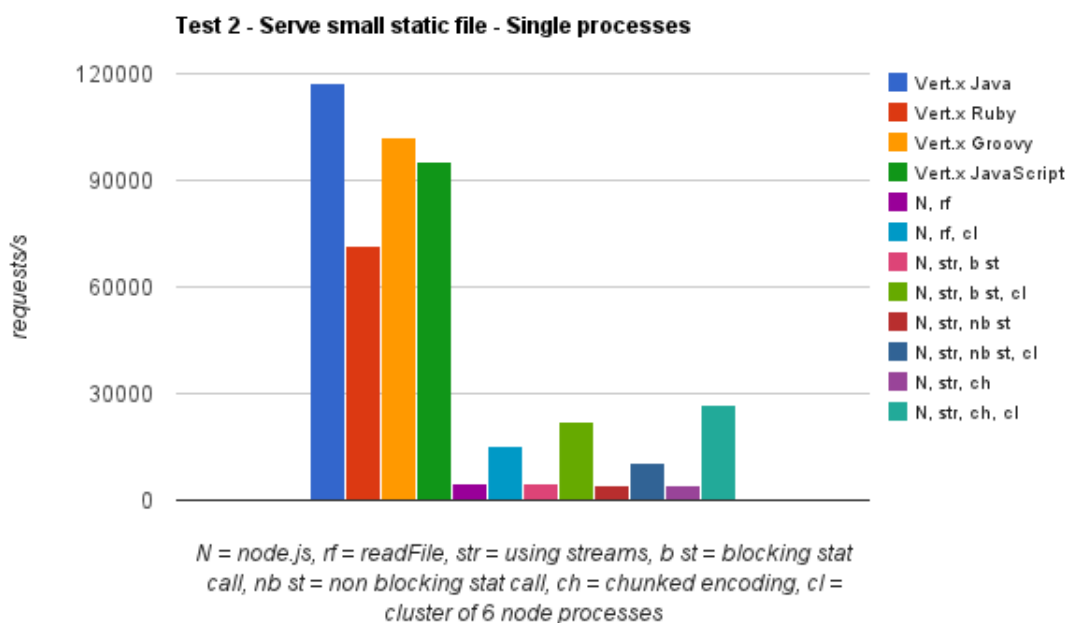
Hardware

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 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 2.6 a 2.7 výsledky obou testů lze shrnout do jedné věty. Vert.x zvládá řádově o desítky tisíc více odpovědí než platforma Node.js a i v případě režimu clusteru.

²⁰HTTP status - OK



Obrázek 2.7: Vert.x instance Výsledek druhého testu Tim Fox

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

2.5.2 Vlastnosti

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

Vlastnost	Node.js	Vert.x
CLI ¹⁸	Ano	Ano
Cluster	Ano	Ano
Moduly	Ano	Ano
HA	Ne	Ano
MQ	Ne	Ano
Hybridní model vláken	Ne	Ano

skytuje 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.

3 Praktická část

popis

3.1 Návrh

Schéma myšlenkové mapy.

```
{
  "_id": "1234-5678-9012-3456",
  "name": "Berries",
  "children": [
    {
      "key": "1",
      "name": "Grapes",
      "children": [
        {
          "key": "2",
          "name": "Red grapes"
        },
        {
          "key": "3",
          "name": "Green grapes"
        }
      ]
    },
    {
      "key": "4",
      "name": "Strawberries"
    },
    {
      "key": "5",
      "name": "Blueberries"
    }
  ]
}
```

3.1.1 Cíle aplikace

- Přidání a odstranění myšlenkových map
- Přidání a odstranění jednotlivých bodů v myšlenkové mapě
- Nevim

3.1.2 Architektura

3.2 Řídící verticle

Vzhledem k tomu, že bude aplikace nasazená na více serverech ve více rolích bylo by potřeba napsat několik modulů podle zaměření, které tam startovat. To lze vyřešit napsáním „startéru“, který dle konfigurace nastartuje příslušné moduly či verticly.

```
JsonObject config = new JsonObject();
config.putString("foo", "wibble");
config.putBoolean("bar", false);
container.deployVerticle("foo.ChildVerticle", config);
```

Spuštění verticle z příkazové řádky

```
vertx run foo.js -conf myconf.json
```

3.3 Integrace s databází MongoDB

databaze

3.4 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 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 3.1. V případě, kdy u uživatele 1 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.

3.4.1 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.

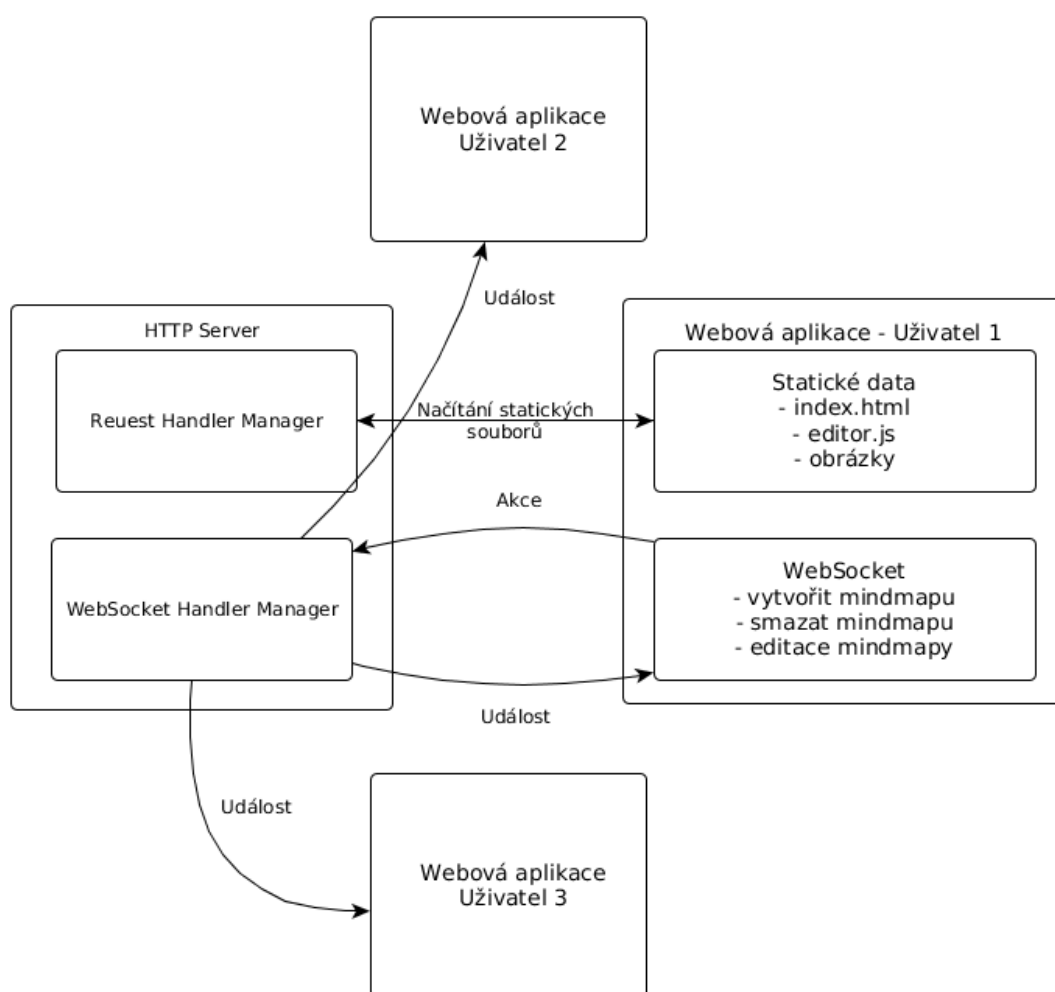
3.4.2 Události

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.

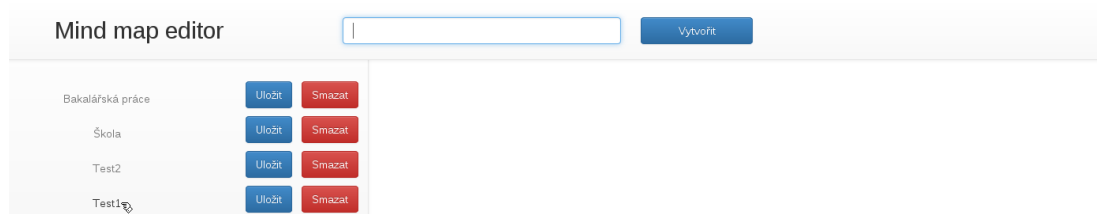
3.5 Polyglot vývoj a moduly

deskriptortoto je poze základní výčet parametrů všechny lze nalézt v dokumentaci Vert.x

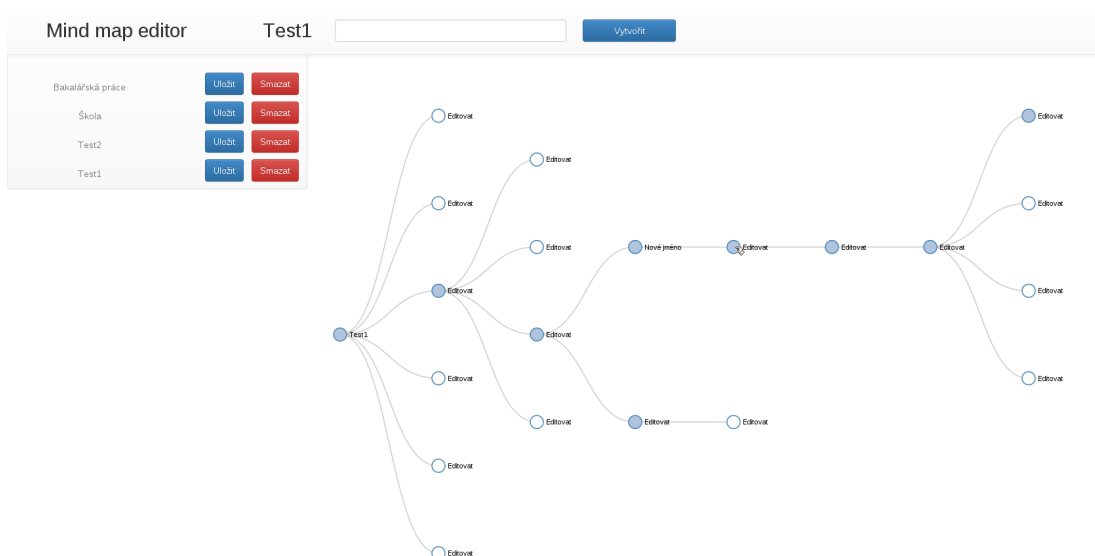
```
{
  "main": "EchoServer.java",
  "worker": true,
  "includes": "io.vertx~some-module~1.1",
  "auto-redeploy": true
}
```



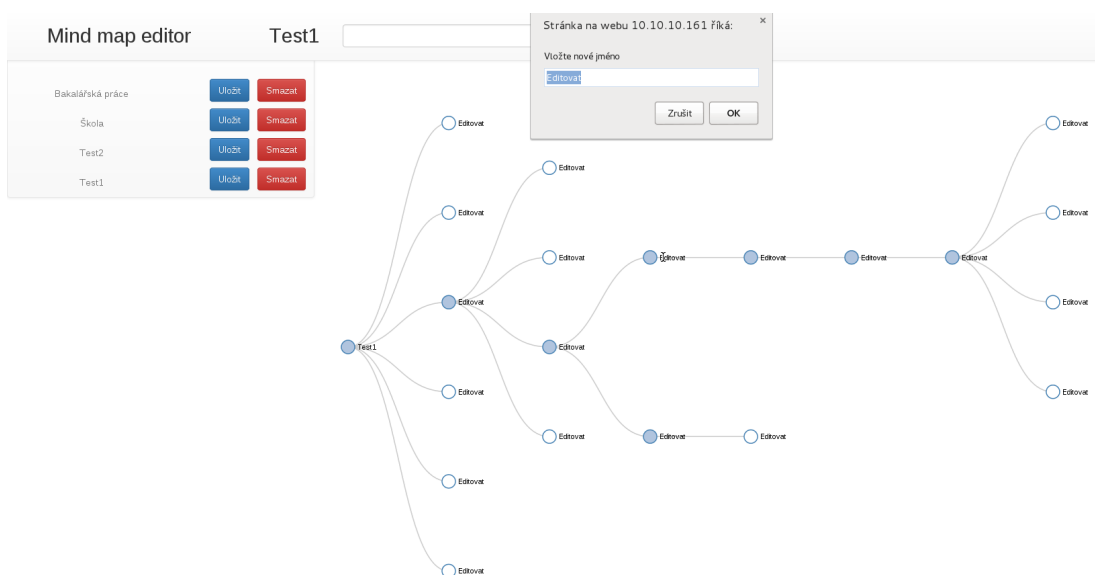
Obrázek 3.1: Komunikace v reálném čase



Obrázek 3.2: Webová aplikace



Obrázek 3.3: Webová aplikace - otevření myšlenkové mapy



Obrázek 3.4: Webová aplikace - akce - editace

Parametr *auto-redeploy* mluví sám za sebe.

Jak bylo řečeno v 2.2.2 Vert.x instance má dvě sady vláken. Parametrem *worker* v deskriptoru modulu, lze říci Vert.x jádru aby spustil modul v *background worker poolu*.

Spuštění modulu programově v jazyce Java

```
container.deployModule("io.vertx~mod-mailer~2.0.0-beta1",  
    JSONconfig);
```

Spuštění modulu z příkazové řádky

```
vertx runmod com.mycompany~my-mod~1.0 -conf config.json
```

moduly více jazyku

3.6 Nasazení

deploy + scaling

3.6.1 Server

ubuntu

3.6.2 Java

java

3.6.3 Vert.x

vert.x

3.6.4 MongoDB

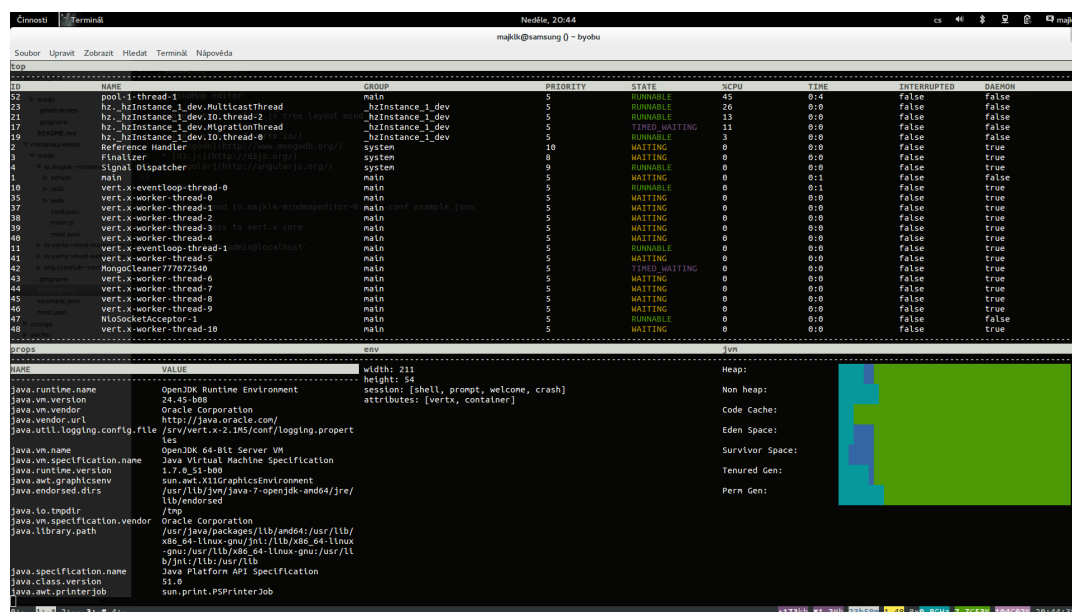
mongodb

3.7 Škálování a vysoká dostupnost

možnosti škálování a HA

3.7.1 Počet Verticlů

verticle count



Obrázek 3.5: Modul CrasHub Shell

3.7.2 Interakce s Vert.x

Díky modulu CrasHub Shell¹ se lze protokolem SSH² 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. Samotný modul pak nabízí jednoduchou možnost přidání vlastním příkazů. Na obrázku je pak vidět přehledová obrazovka. Na obrazovce je seznam všech vláken, celkové velikost zásobníku či verze použité Javy.

3.7.3 Vert.x v clusteru

Pro zajištění bezpečnosti databázového serveru, kde běží také služba pro vzdálený přístup k Vert.x instanci bude aplikace nasazena v clusteru. Webový server(HTTP Server) na obrázku 3.6 je připojen do dvou sítí.

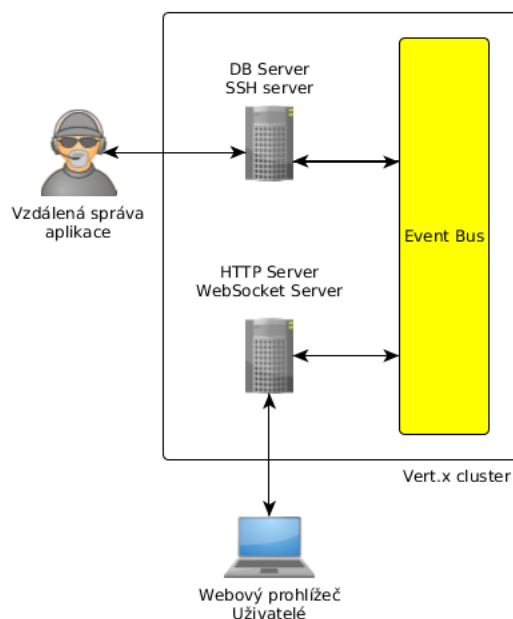
Internet pouze webový server

Síť clusteru všechny servery

Pro maximální bezpečnost budou na webovém serveru otevřeny pouze porty 80 a 443. Na druhém serveru pak poběží samotná databáze a druhá část aplikace, která v

¹<https://github.com/crashub/mod-shell>

²Secure Shell



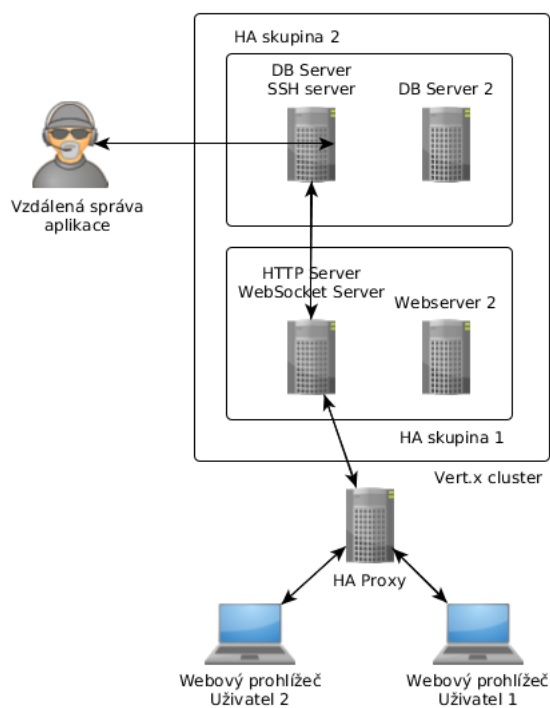
Obrázek 3.6: Architektura nasazené aplikace

sobě bude zahrnovat modul zajišťující komunikaci s databází a modul pro vzdálenou správu Vert.x instancí, který byl představen v kapitole 3.7.2.

3.7.4 Vysoká dostupnost

Pro zajištění vysoké dostupnosti klíčových prvků aplikace, je potřeba upravit architekturu clusteru. Před webový server je postavená HA proxy³, která při úpadku jednoho z webových serverů přeměruje komunikaci na server druhý. Vert.x cluster je pak rozdělený na dvě HA skupiny (obr.3.7), 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 ležet nemusí.

³služba zajišťující vyrovnavání zatížení



Obrázek 3.7: Ideální architektura nasazení aplikace

4 Závěr

Podářilo se vytvořit webovou aplikaci, do které jdou jednoduše přidávat a odebírat mindmapy. Pokud má stejnou mindmapu otevřeno více lidí najednou, okamžitě vidí všechny změny, ostatních klientů. Aplikace používá volně šiřitelný software, který je ve většině případů špičkové úrovni. Tento postup dovoluje, s relativně nízkými náklady, řešit velmi komplikované problémy. 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 mindmap či případné sdílení jednotlivých mindmap.

Literatura

- [1] Phipps, Simon *Who controls Vert.x: Red Hat, VMware, or neither?* [online]. [cit. 2014-06-30]. Dostupný z WWW: <http://www.infoworld.com/d/open-source-software/who-controls-vertx-red-hat-vmware-or-neither-210549>
- [2] Kamali, Masoud *The Winners of the JAX Innovation Awards 2014* [online]. [cit. 2014-06-30]. Dostupný z WWW: <http://jax.de/awards2014/>
- [3] 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/>
- [4] Merta, Zdeněk *Vert.x jOpenSpace 2013* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://jopenspace.cz/2013/presentations/zdenek-merta-vert.x.pdf>
- [5] 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>
- [6] *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
- [7] 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/>
- [8] 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/>
- [9] Pitner, Tomáš *Programování v jazyce Java* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.fi.muni.cz/~tomp/slides/pb162/printable.html>

-
- [10] Lažanský, J. *Procesy a vlákna* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://labe.felk.cvut.cz/vyuka/A4B33OSS/Tema-03-ProcesyVlakna.pdf>
- [11] Fox, Tim *Event loops* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://vertx.io/manual.html#event-loops>
- [12] Kosek, Jiří *Session proměnné* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.kosek.cz/clanky/php4/session.html>
- [13] Janssen, Cory *Message Queue* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.techopedia.com/definition/25971/message-queue>
- [14] Froemke, Dina *Framework Benchmarks Round 8* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.techempower.com/blog/2013/12/17/framework-benchmarks-round-8/>
- [15] Fox, Tim *Vert.x vs node.js simple HTTP benchmarks* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://vertxproject.wordpress.com/2012/05/09/vert-x-vs-node-js-simple-http-benchmarks/>

Přílohy

Seznam obrázků

2.1	Architektura Vert.x <i>Jaehong Kim</i>	4
2.2	Vert.x instance	8
2.3	Vert.x instance <i>vertx run HelloWorld -instances 4</i>	9
2.4	Event Bus distribuovaný mezi dva servery	11
2.5	Clustering mezi třemi Vert.x instancemi	15
2.6	Výsledky druhého testu <i>Tim Fox</i>	17
2.7	Vert.x instance <i>Výsledek druhého testu Tim Fox</i>	18
3.1	Komunikace v reálném čase	23
3.2	Webová aplikace	23
3.3	Webová aplikace - otevření myšlenkové mapy	24
3.4	Webová aplikace - akce - editace	24
3.5	Modul CrasHub Shell	26
3.6	Architektura nasazené aplikace	27
3.7	Ideální architektura nasazení aplikace	28

Seznam tabulek

2.1	Srovnání odezvy	18
2.2	Srovnání vlastností s Node.js	18