

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 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 22. července 2014

Michael Kutý

# Obsah

<b>Abstrakt</b>	<b>ii</b>
<b>Zadání práce</b>	<b>iii</b>
<b>1. Úvod</b>	<b>1</b>
1.1. Cíl a metodika práce . . . . .	1
1.2. Postup a předpoklady práce . . . . .	2
<b>2. Platforma Vert.x</b>	<b>3</b>
2.1. Historie . . . . .	3
2.2. Architektura . . . . .	4
2.2.1. Jádro . . . . .	4
2.2.2. API . . . . .	5
2.2.3. Multi-reactor pattern . . . . .	6
2.2.4. Vert.x instance . . . . .	7
2.2.5. Event Bus . . . . .	10
2.2.6. Hazelcast . . . . .	11
2.3. Test . . . . .	12
<b>3. Praktická část</b>	<b>14</b>
3.1. Návrh . . . . .	14
3.1.1. Cíle aplikace . . . . .	14
3.2. Základní aplikace . . . . .	14
3.3. Integrace s databází MongoDB . . . . .	14
3.4. Real-time komunikace . . . . .	14
3.5. Polygnot vývoj a moduly . . . . .	14
3.6. Nasazení . . . . .	14
3.6.1. Server . . . . .	14
3.6.2. Java . . . . .	14
3.6.3. Vert.x . . . . .	15
3.6.4. MongoDB . . . . .	15
3.7. Škálování a vysoká dostupnost . . . . .	15
3.7.1. Počet Verticlů . . . . .	15
3.7.2. Vert.x v clusteru . . . . .	15
<b>4. Závěr</b>	<b>16</b>
<b>Literatura</b>	<b>17</b>
<b>Přílohy</b>	<b>I</b>
<b>A. Seznam obrázků</b>	<b>II</b>



# 1. Úvod

V současné době existuje nespočet frameworků<sup>1</sup> 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 polygnot programování s jednoduchým konkurenčním modelem a možnost sdílet data mezi jednotlivými vlákny bez nutnosti zámků.

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.js<sup>2</sup> 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.

---

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

<sup>2</sup>Serverový framework, postavený na modelu událostmi řízeného programování

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 real-time webových aplikací
- MongoDB: dokumentové orientovaná NoSQL databáze
- AngularJS: client side framework pro snadný a efektivní vývoj jednostránkových webových aplikací
- D3.js: framework pro práci s grafy

## 2. Platforma Vert.x

Dnešním trendem internetu jsou real-time kolaborativní aplikace, které drasticky změnily 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 C10K<sup>1</sup> problém. Platforma Vert.x má velice podobné API<sup>2</sup> 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].

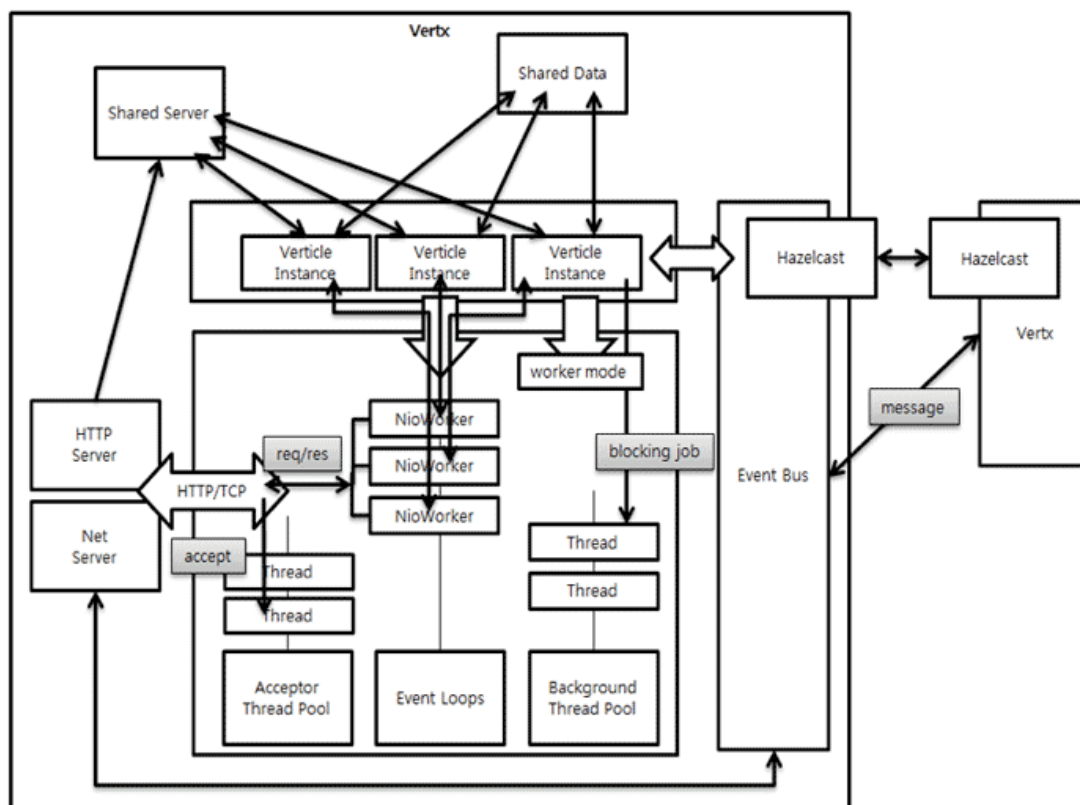
Po několika debatách jak s představiteli společnosti RedHat tak i komunitou došel Tim Fox k názoru, že nejlepší pro budoucí zdravý rozvoj platformy bude přesunutí celé platformy pod nadaci Eclipse Foundation, k čemuž došlo na konci roku 2013. V dnešní době se platforma těší velkému vývoji, který čítá 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].

---

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

<sup>2</sup>Application Programming Interface

Obrázek 2.1.: Architektura Vert.x *Jaehong Kim*

## 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 stabilní API. Lze jej následně rozšířit o novou funkčnost dokompilováním balíčků, které lze nalézt v oficiálním repositáři. Pravděpodobnou inspirací byl již zmíněný Node.js respektive NPM<sup>3</sup> 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. Událostmi řízené programování je podle Tomáše Pitnera[8] základním principem tvorby aplikací s GUI(Graphical user interface). Netýká se však pouze GUI, je to obecnější pojem označující typ asynchronního programování, kdy je:

<sup>3</sup>Node package manager

tok programu řízen událostmi; 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 Vlákno<sup>4</sup> je posloupnost po sobě jdoucích událostí.(vlákno). 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 a jednotlivé úlohy co při běhu aplikace vznikají si řídí sám.

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[9] 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`<sup>5</sup>.

**ExecutorService** z balíčku `java.util.concurrent`

**CyclicBarrier**<sup>6</sup> z balíčku `java.util.concurrent`

**CountDownLatch** z balíčku `java.util.concurrent`

**File** z balíčku `java.nio`

**Vylepšený ClassLoader** lepší odolnost vůči deadlockům<sup>7</sup>

*Více o `java.concurrent`[5]*

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`<sup>8</sup>.

## 2.2.2. 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

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

<sup>5</sup>Knihovna pro práci s multitaskingem

<sup>7</sup>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.

<sup>8</sup><http://www.oracle.com/technetwork/articles/java/fork-join-422606.html>

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 se dělí na *Základní API* a *Kontainer API*.

### 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 2.2.4.

- 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

### 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 CLI<sup>9</sup>.

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

#### 2.2.3. Multi-reactor pattern

Základ jádra je postaven na tzv. Multi-reactor pattern[10], který vychází z Reactor patternu[4], ten lze charakterizovat několika body:

- aplikace je řízena událostmi
- na události se registrují handlers

---

<sup>9</sup>Command Line Interface

- vlákno zpracovává události a spouští registrované handlersy
- toto vlákno nesmí být blokováno<sup>10</sup>

Multi-reactor pattern[10] 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. Takovému 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 následující straně pak lze vidět situaci čtyř hlavních vláken na čtyři procesorové jádra. ?? na straně ??říklady 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 2.2.4

#### 2.2.4. Vert.x instance

Verticle běží v jedné Vert.x instanci 2.2 na následující straně. Každá Vert.x instance běží ve vlastním JVM instanci. V jedné Vert.x instanci může najednou běžet X Vertclů. Na jednom fyzickém stroji může běžet více Vert.x instancí případně v cluster módu i na více fyzických strojích.

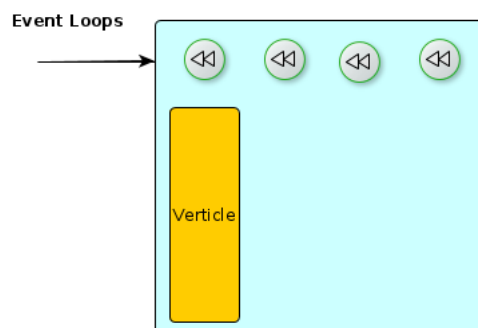
### Verticle

Základní jednotka vývoje a nasazení. Verticle může být skript nebo třída například v jazyce Java. Verticle lze spouštět samostatně<sup>11</sup> v praxi se ovšem využívají pouze moduly, které obsahují zpravidla více Verticles popřípadě worker Verticles.

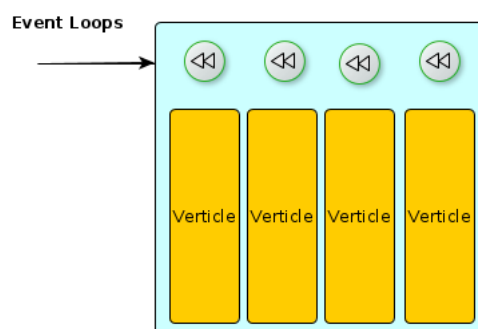
- nejmenší spustitelná jednotka
- třída / skript

<sup>10</sup> pokud dojde k zablkování hlavního vlákna dojde k zablokování celé aplikace např. `Thread.sleep()`, a další z `java.util.concurrent`

<sup>11</sup> `vertx run Verticle.js`



Obrázek 2.2.: Vert.x instance

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



- vykonává neblokující operace
- konkurence - single-threaded<sup>12</sup>
- přístup ke Core API 2.2.2, registrace handlerů, deploy dalších verticlů

#### Spuštění verticle programově

```
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
```

## Moduly

Moduly poskytují větší míru zapouzdření a znovupoužitelnost funkcionality. V praxi se mohou moduly skládat z více modulů či verticlů a mohou být uloženy v centrálním repozitáři<sup>13</sup> 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 JSON<sup>14</sup>, tento deskriptor musí být v kořenovém adresáři modulu a může vypadat například takto. *toto je pouze 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
}
```

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

**spustitelné** mají definovanou main třídu v deskriptoru, takovéto moduly je pak možné spustit jako samostatné jednotky pomocí parametru *runmod* nebo *programově deploy-Module*

**nespustitelné** modul nemá specifikovanou main třídu a lze jej použít v jiném modulu použitím parametru *includes*

Jak bylo řečeno v 2.2.3 Vert.x instance má dvě sady vláken. Parametrem *worker* v deskriptoru modulu, lze říci Vert.x jádru aby spustil modul v *background worker poolu*. Parametr *auto-redeploy* mluví sám za sebe.

<sup>12</sup>běží vždy pouze v jednom vlákně (odpadá synchronizace, zámky, ...), izolace (vlastní classloader)

<sup>13</sup><http://modulereg.vertx.io/>

<sup>14</sup>JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojem.

### 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
```

## Worker Verticle

### 2.2.5. Event Bus

Nervový systém celého Vert.x. Cílem EventBusu je zpozdředkování komunikace mezi jednotlivými komponentami platformy. Nespornou výhodou je fakt, že lze takovouto komunikaci přemostit ke klientovi na straně webového prolížeče.

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 jsou pouze základní typy zpráv, které Vert.x podporuje v základu. Není ale vůbec problém výčet stávajících typů rozšířit (doimplementovat). Například modul `bson.vertx.eventbus`<sup>15</sup> rozšíří aplikaci o možnost používat mnohem komplexnější typy zpráv. Mezi doporučené se ovšem řadí JSON, protože je jednoduše serializovatelný mezi jednotlivými programovacími jazyky.

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

<sup>15</sup><https://github.com/pmlopes/mod-bson-io>

## 2.2.6. Hazelcast

Jednou z nejdůležitějších architektonických součástí Vert.x je knihovna Hazelcast<sup>16</sup>, Hlavní výhody In-memory data grid[6] lze podle Ki Sun Song sumarizovat:

- Data jsou distribuovaná a uložena na více serverech
- 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:

- In-memory NoSQL<sup>17</sup>
- Caching<sup>18</sup>
- Data grid
- Messaging
- Application Scaling
- Clustering

Hazelcast je tedy typ distribuovaného úložiště, které běží jako embedded a lze díky němu distribuovat celou aplikaci. Hazelcast API je využíváno přes API Vert.x. Když je Vert.x spuštěn, Hazelcast je spuštěn v embedded<sup>19</sup> módu. Jako nejčastější příklad bývá uváděno ukládání uživatelské session<sup>20</sup> Hazelcast tedy usnadní práci v situaci, kdy budeme potřebovat uložit uživatelskou session například pro eshop. Mohli bychom využít využít externí RDBMS<sup>21</sup> díky, kterému by jsme dosáhli stejného výsledku. Hazelcast nám ovšem zaručí replikování mezi jednotlivými servery, fail-over S využitím embedded Hazelcast ovšem odpadá nezbytná režie a monitoring, nemluvě o serverových prostředcích.

Proto ty, kteří potřebují ukládat uživatelské session pro E-commerce či chat-servery toho mohou jednoduše dosáhnout skrz konfiguraci samotného Vert.x.

<sup>16</sup>okolo 2.6MB kódu v jazyce Java, In-Memory Data Grid (IMDG)

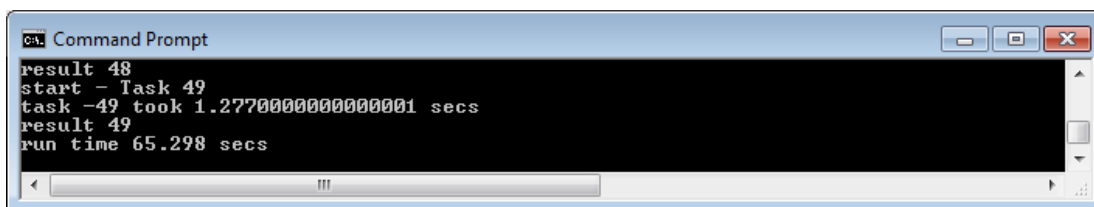
<sup>17</sup>databázový koncept, ve kterém datové úložiště i zpracování dat používají jiné prostředky než tabulková schémata tradiční relační databáze

<sup>18</sup>specializovaný typ paměti pro krátkodobé ukládání

<sup>19</sup>Hazelcast server je spuštěn jádrem Vert.x

<sup>20</sup>Session v protokolu HTTP dává webovému serveru možnost uložit si libovolné (většinou však ne příliš obsáhlé) informace o uživateli, kteří k němu přistupují, a to o každém zvlášť. Protokol HTTP ze svého principu (a způsobu komunikace stylem požadavek - odpověď) postrádá kontext o jednotlivých klientech, a právě session ho webovým aplikacím dokáže dát.

<sup>21</sup>Databázový server, který spravuje databáze, komunikaci s klienty (lokálními nebo vzdálenými), vstupy a výstupy dat a jejich integritu.

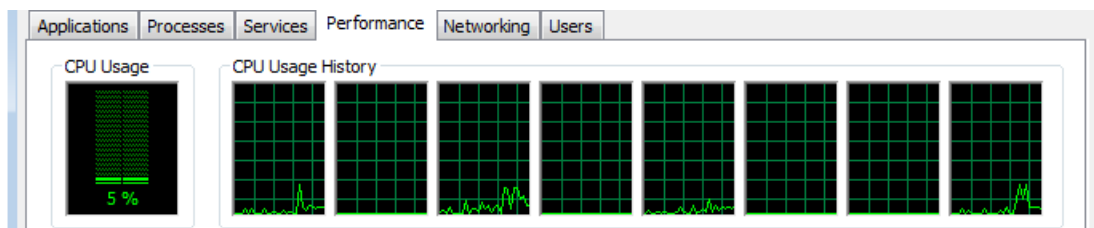


```

CA: Command Prompt
result 48
start - Task 49
task -49 took 1.2770000000000001 secs
result 49
run time 65.298 secs

```

Obrázek 2.4.: První test běhu serializační třídy



Obrázek 2.5.: Využití jednotlivých procesorů při běhu

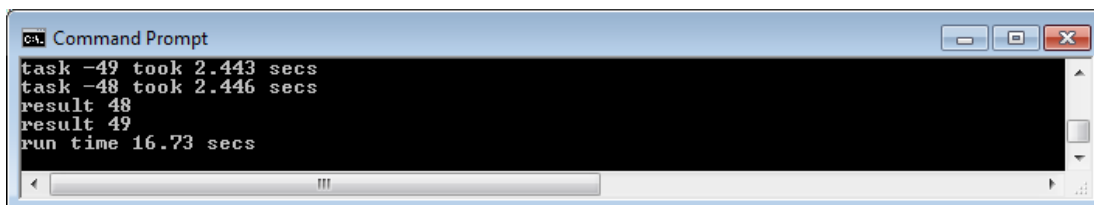
## 2.3. Test

Ed Gardoh v roce 2011 provedl test[3] pro porovnání paralelizace<sup>22</sup> v Javě 1.6 a 1.7. Hlavní myšlenkou je aby testovací třída simulovala úkol, který jako první volá vzdálenou službu a čeká sekundu na výzvu k návratu(spánek) a pak simuluje nějaké zpracování s výsledkem, jako je formátování řetězce. 2.4 je vidět synchronní běh serializační třídy v Javě 1.6. Z 2.5 je pak vidět využití potenciálů jednotlivých procesorů. Výsledek není žádné překvapení 50 úkolů s 1 sekundovým spánkem a spojováním řetězce trvalo něco málo přes 65 sekund. Cílem jeho testu mělo být porovnání paralelizování úkonů. Výsledky testu ukázaly zlepšení až o 75%. Z obrázků 1-4 zřetelně plyne, že nová Java je, pro single-thread<sup>23</sup> model aplikace ta nejlepší volba.

Jetnotlivé testy prokázaly, že za takovým rapidním zrychlením stojí metody Fork/Join. Při vhodném škálování bylo zrychlení až o 75%. Z testů ovšem vyplývá také fakt, že při neúměrném počtu hlavních vláken na počet procesorů to má negativní dopady. Jedním z dopadů je 100% vytížení a jednotlivých jader. Při vhodném určení počtu vláken, je vidět rapidní urychlení asynchronní paralelizace. Node.js i Vert.x však poskytují informace o celkovém počtu fyzických jader procesoru a ta je tedy snadné určení optimálního počtu vláken pro ideální výsledky.(Více na?asi vysvětlit)

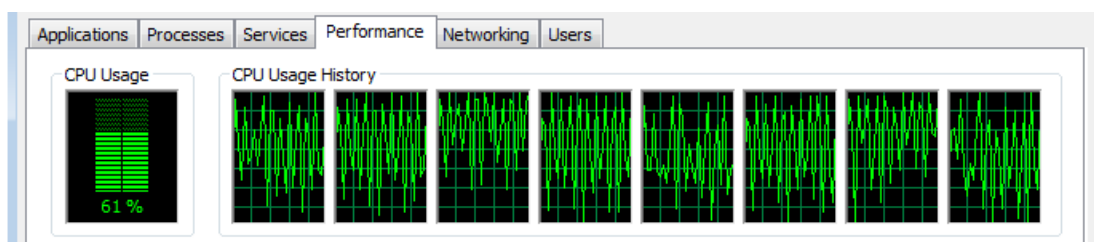
<sup>22</sup>Paralelizace procesů se skládá z rozložení jednoho velkého úkonu do několika menších úkolů, které mohou běžet paralelně. Výsledkem je provedení jednoho úkonu nebo procesu za pomoci více než jednoho procesoru nebo procesorů "Paralelní zpracování", nesmí být zaměňováno se souběžností.

<sup>23</sup>jedno vláknový



```
cmd Command Prompt
task -49 took 2.443 secs
task -48 took 2.446 secs
result 48
result 49
run time 16.73 secs
```

Obrázek 2.6.: První test běhu serializační třídy



Obrázek 2.7.: První test běhu serializační třídy

## 3. Praktická část

popis

### 3.1. Návrh

test

#### 3.1.1. Cíle aplikace

- Přidání a odstranění jednotlivých bodů v MindMapě

### 3.2. Základní aplikace

core

### 3.3. Integrace s databází MongoDB

databaze

### 3.4. Real-time komunikace

komunikace

### 3.5. Polygnot vývoj a moduly

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

### **3.7.2. Vert.x v clusteru**

HA

## 4. Dobrá rada na závěr

LyX je vynikající editor, který vám usnadní napsání rozsáhlejší práce typu bakalářka nebo diplomka. Editor si hravě poradí s komplikovanými úlohami jako je vkládání křížových odkazů, vytvoření seznamu literatury a citování literatury v textu, vytvoření obsahu a rejstříku. Bez většího úsilí bude vaše práce typograficky na úrovni.

Používáte-li LyX jen na psaní bakalářky, *nesnažte se* naučit vše, co umí! Zabralo by to více času než celá bakalářka! Naučte se jen pár nezbytností a pište a pište a pište! Až budete mít dopsán a zkontrolován text, můžete si pohrát s výběrem vzhledu vhodného pro vaši práci, s výběrem písma, typu záhlaví stránek, hlaviček kapitol atd. Teprve nakonec udělejte závěrečnou typografickou revizi textu. Zejména zkontrolujte polohu plovoucích objektů (případně je přemístěte na vhodnější místo) a odstraňte vdovy a sirotky (osamělé řádky)<sup>1</sup>.

!

---

<sup>1</sup>Nejsnáze odstranit tak, že z textu vypustíte (nebo do něj přidáte) pár slov či vět anebo úpravou odstavců.



# 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] *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](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html#package_description)
- [6] 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/>
- [7] 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/>
- [8] Pitner, Tomáš *Programování v jazyce Java* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.fi.muni.cz/~tomp/slides/pb162/printable.html>
- [9] Lažanský, J. *Procesy a vlákna* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://labe.felk.cvut.cz/vyuka/A4B33OSS/Tema-03-ProcesyVlakna.pdf>
- [10] Fox, Tim *Event loops* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://vertx.io/manual.html#event-loops>

# **Přílohy**

## A. Seznam obrázků

Klávesová zkratka Alt+p (Paragraph style – výběr stylu odstavce)		
Zkratka	Styl	Mnemo
Alt+p 0	Část	Pořadí úrovně
Alt+p 1	Kapitola	dtto
Alt+p 2	Sekce	dtto
Alt+p 3	Podsekce	dtto
Alt+p 4	Podpodsekce	dtto
Alt+p 5	Odstavec	dtto
Alt+p 6	Pododstavec	dtto
Alt+p s	Standardní	
Alt+p n nebo e	Číslovaný seznam	Numeric (Enumerate)
Alt+p b nebo i	Seznam	Bulleted (Item)
Alt+p q	Citát	Quotation
Alt+p d	Popis	Description
Další zkratky		
Zkratka	Činnost	Poznámka
Ctrl+c	kopírovat	
Ctrl+v	vložit	
Ctrl+x	vyjmout	
Ctrl+e	vybraný text zdůraznit	obvykle kurzíva
Ctrl+b	vybraný text tučně	<b>Bold</b>
Ctrl+u	vybraný text podtržené	<u>Underline</u>
Ctrl+mínus	doporučené místo dělení slova	
Ctrl+mezerník	nezlomitelná mezera	
Ctrl+s	uložit dokument	
Ctrl+z	zpět	

Alt+s (Font size – změna velikosti písma)		
Alt+s s	malé	Small
Alt+s n	normální	Normal
Alt+s l	velké	large
Alt+s Shift+L	větší	Large
Alt+s h	největší	huge
Alt+s Shift+H	obrovské	Huge
Alt+s plus	větší	
Alt+s mínus	menší	
Alt+c (Character style – změna stylu písma)		
Alt+c r	patkové písmo	Roman
Alt+c s	bezpatkové písmo	Sans
Alt+c p	(kód) strojopis	Program
Alt+c c	kapitálky	Capitals
Alt+c ↑	všechna velká	
Alt+c ↓	všechna malá	
Alt+c →	první velká	
Alt+c mezerník	základní písmo	
Matematické výrazy		
Ctrl+m	vložit matematický výraz v původní řádce	
Ctrl+Shift+M	vložit matematický výraz na nové řádce	
Ctrl+mínus	→	
Příkazy platné v matematickém režimu		
_ (podtržítko)	dolní index	
^	horní index	