

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 do 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 30. června 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 . . . . .	1
<b>2. Platforma Vert.x</b>	<b>1</b>
2.1. Architektura . . . . .	1
2.1.1. Jádro . . . . .	1
2.1.2. Vert.x instance . . . . .	5
2.1.3. Event Bus . . . . .	7
2.1.4. Hazelcast . . . . .	8
2.2. Test . . . . .	9
<b>3. Praktická část</b>	<b>1</b>
3.1. Návrh . . . . .	1
3.2. Realizace . . . . .	1
<b>4. Závěr</b>	<b>2</b>
<b>Literatura</b>	<b>4</b>
<b>Přílohy</b>	<b>I</b>
<b>A. Přehled klávesových zkratk v Lyxu</b>	<b>II</b>



## 1.1. Cíl a metodika práce

Hlavním cílem práce bude zjištění jestli platforma Vert.x splňuje všechny předpoklady moderní platformy pod kterou lze vyvíjet distribuovanou single-page aplikaci dále jen SPA.

Hlavním cílem práce bude zjištění jestli se platforma Vert.x hodí pro vývoj distribuovaných single-page aplikací dále jen SPA. Čtenáři a vytvoření jednoduchého webového mindmap editoru. Jednostránkové webové aplikace pro kolaborativní práci s mindmapami. Na této jednoduché aplikaci bude demonstrován celý 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 programováním a jednoduchým konkurenčním modelem. Cílem teoretické části je tedy popsat jednotlivé části platformy a jejich účel či problém, který řeší. V závěru teoretické části bude platforma srovnána s několika významnými frameworky a to v několika důležitých aspektech rychlosti, která je v dnešním světě neustálého růstu počtu zařízení, je to co trápí webové aplikace s desítkami tisíc připojených klientů.

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 pracovní stanice, kde bude prověřena funkčnost distribuovaného provozu aplikace. Pro nasazení aplikace na více pracovních stanic bude použit nástroj konfiguračního managementu Salt Stack.

## 1.2. Postup a předpoklady práce

Práce předpokládá základní znalost programovacího jazyku Java. Teoretická část se neomezuje pouze na nezbytný popis technologií potřebných k realizaci malé jednostrán-

kové webové aplikaci. 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

## PLATFORMA VERT.X

Dnešním trendem jsou real-time webové aplikace. Pro vývoj kolaborativní aplikací jsou již svěřené techniky a postupy, které implementuje například Node.js či ruby EventMachine. Problém těchto jinak časem a komunitou ověřených platforem je jazyk, v kterém jsou implementovány. Obě zmíněné platformy jsou napsány v dynamicky kompilovaném jazyku, což pro jádro stabilní aplikace přináší povinnost psát jak testy integrační, které testují funkčnost celého systému, tak i unit testy. I když bude aplikace z větší části pokryta testy, mohou se objevit problémy v podobě nečekaných pádů za běhu aplikace. To může být způsobeno například voláním neexistující metody či přiřazení proměnné do jiného typu než je ona sama. Toto bylo jedním z důvodů pro implementaci nového řešení v jazyce Java. Tento jazyk přináší platformě velkou stabilitu, rozšiřitelnost a zázemí v podobě tisícovek stabilních knihoven. Vert.x může být použit jako plnohodnotné řešení pro celou aplikaci nebo nasazen jako dílčí část architektury jiného řešení.

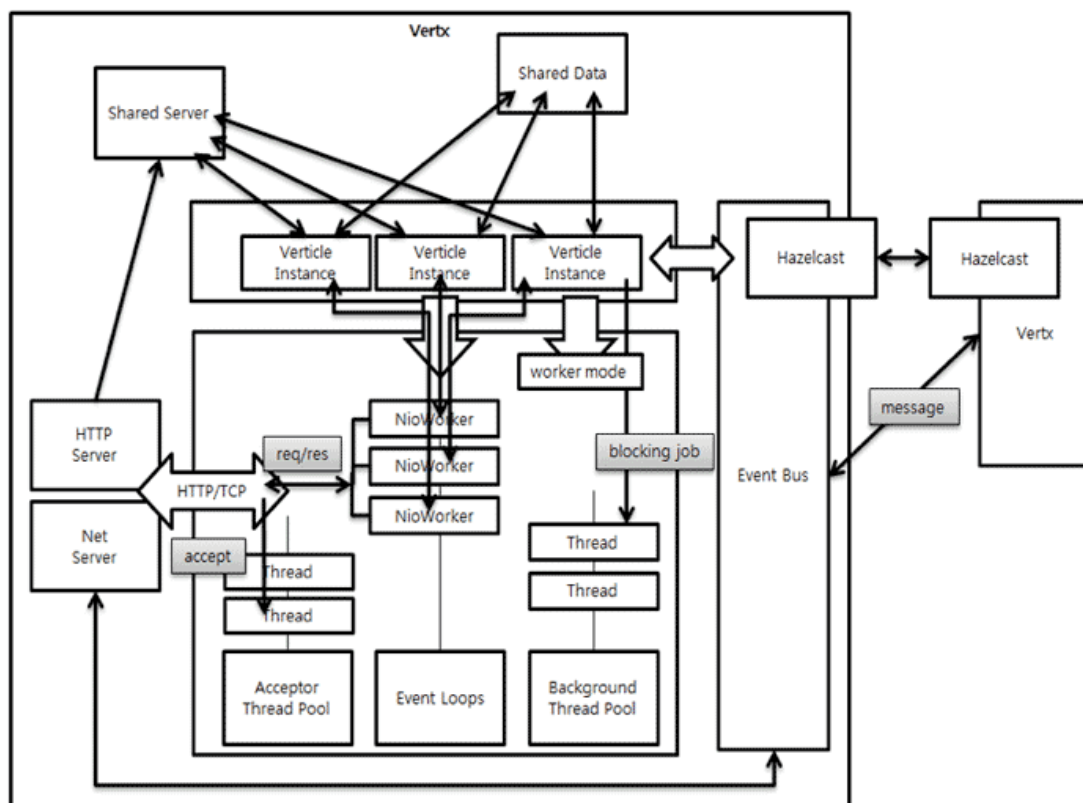
## 2.1. Architektura

### 2.1.1. Jádro

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 naléznout v oficiálním repozitáři. Pravděpodobnou inspirací byl již zmíněný Node.js respektive NPM<sup>1</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[6] 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; události nastávají obvykle určitou uživatelskou akcí:

---

<sup>1</sup>Node package manager



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

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>2</sup> je posloupnost po sobě jdoucích událostí.(vlákno). V dřívější době nebylo potřeba rozlišovat proces a vlákno, protože proces se dále v aplikaci nedělil. Základem Vert.x respektive Node.js je tedy vícevláknový model. V jedné aplikaci tedy může běžet několik vláken. Vlákno je zde bráno jako základní plánovací jednotka pro běh na procesoru. 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[7] 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 zajímá nás tedy jak moc je dobrá implementace paralelního modelu. Zde se dostáváme k jedinému požadavku pro běh Vert.x instancí a to je přítomnost Java development Kitu ve verzi 1.7. 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>3</sup>.

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

**CyclicBarrier**<sup>4</sup> z balíčku `java.util.concurrent`

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

**File** z balíčku `java.nio`

**Vylepšený ClassLoader** lepší odolnost vůči deadlockům<sup>5</sup>

*Více o `java.concurrent`[3]*

Ed Gardoh v roce 2011 ve svém jednoduchém testu[1] 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>6</sup>.

## základní API

- TCP/SSL server/klient
- Websockets server/klient, SockJS
- Event Bus / sdílená data
- časovače

<sup>2</sup>Označuje v informatice odlehčený proces, pomocí něhož se snižuje režie operačního systému při změně kontextu, které je nutné pro zajištění multitaskingu

<sup>3</sup>Knihovna pro práci s multitaskingem

<sup>5</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>6</sup><http://www.oracle.com/technetwork/articles/java/fork-join-422606.html>

- souborový systém
- konfigurace
- logování

### Multi-reactor pattern

Základ jádra je postaven na tzv. Multi-reactor pattern[8], který vychází z Reactor patternu[2], ten lze charakterizovat několika body:

- aplikace je řízena událostmi
- na události se registrují handlers
- vlákno zpracovává události a spouští registrované handlers
- toto vlákno nesmí být blokováno<sup>7</sup>

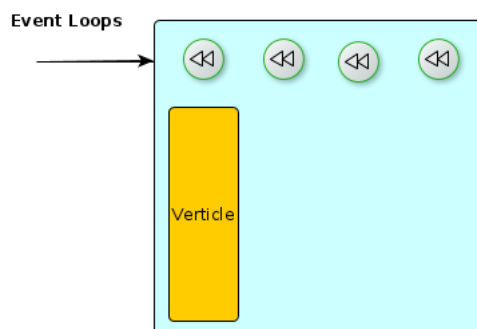
Multi-reactor pattern[8] se od Reactor patternu liší pouze tím, že může mít více hlavních vláken. Hlavní vlákno, kterému se okolo Vert.x komunity říká *Event Loop*. V komunitách Nginx nebo Node.js se ovšem setkáme s pojmem *Run Loop*. Tento návrhový vzor tedy převzala platforma z Node.js, kde se takovýto model velice oblíbil. 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()` na obrázku 2.3 na následující straně lze vidět příklad čtyř hlavních vláken na čtyři dostupné procesory. ?? na straně ?? příklady blokujících volání:

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

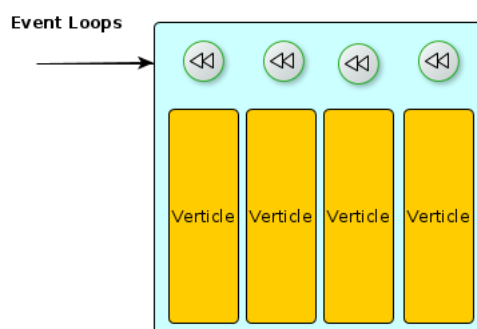
### Hybridní model vláken

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 2. 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 2.1.2

<sup>7</sup> pokud dojde k zablkování hlavního vlákna dojde k zablokování celé aplikace např. `Thread.sleep()`, a další z `java.util.concurrent`



Obrázek 2.2.: Vert.x instance

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

### 2.1.2. Vert.x instance

Verticle běží v jedné Vert.x instanci 2.2. Každá Vert.x instance běží ve vlastním JVM instanci. V jedné Vert.x instanci může najednou běžet X Verticlů. 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>8</sup> v praxi se ovšem využívají pouze moduly, které obsahují zpravidla více Verticles popřípadě worker Verticles.

- nejmenší spustitelná jednotka
- třída / skript
- vykonává neblokující operace
- konkurence - single-threaded<sup>9</sup>

<sup>8</sup>vertx run Verticle.js

<sup>9</sup>běží vždy pouze v jednom vlákně (odpadá synchronizace, zámky, ...), izolace (vlastní classloader)

- přístup ke Core API 2.1.1, 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>10</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>11</sup>, tento deskriptor musí být v kořenovém adresáři modulu a může vypadat například takto. *toto 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
}
```

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 *deployModule*

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

Jak bylo řečeno v 2.1.1 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.

Spuštění modulu programově v jazyce Java

<sup>10</sup><http://modulereg.vertx.io/>

<sup>11</sup>JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojem.



---

```
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

### 2.1.3. Event Bus

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

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>12</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>12</sup><https://github.com/pmlopes/mod-bson-io>

### 2.1.4. Hazelcast

Jednou z nejdůležitějších architektonických součástí Vert.x je knihovna Hazelcast<sup>13</sup>, Hlavní výhody In-memory data grid[4] 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>14</sup>
- Caching<sup>15</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>16</sup> módu. Jako nejčastější příklad bývá uváděno ukládání uživatelské session<sup>17</sup> Hazelcast tedy usnadní práci v situaci, kdy budeme potřebovat uložit uživatelskou session například pro eshop. Mohli bychom využít využít externí RDBMS<sup>18</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>13</sup>okolo 2.6MB kódu v jazyce Java, In-Memory Data Grid (IMDG)

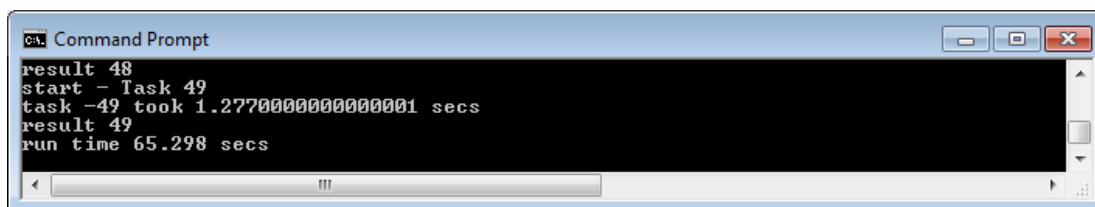
<sup>14</sup>databázový koncept, ve kterém datové úložiště i zpracování dat používají jiné prostředky než tabulková schémata tradiční relační databáze

<sup>15</sup>specializovaný typ paměti pro krátkodobé ukládání

<sup>16</sup>Hazelcast server je spuštěn jádrem Vert.x

<sup>17</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>18</sup>Databázový server, který spravuje databáze, komunikaci s klienty (lokálními nebo vzdálenými), vstupy a výstupy dat a jejich integritu.

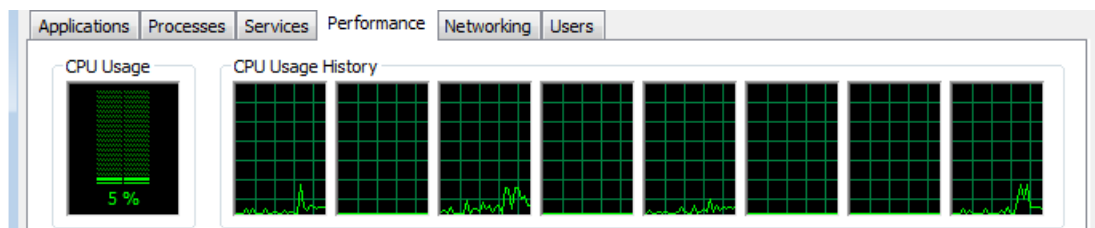


```

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

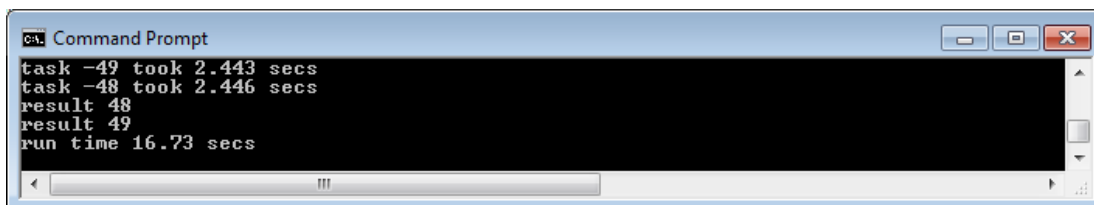
## 2.2. Test

Ed Gardoh v roce 2011 provedl test[1] pro porovnání paralelizace<sup>19</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>20</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)

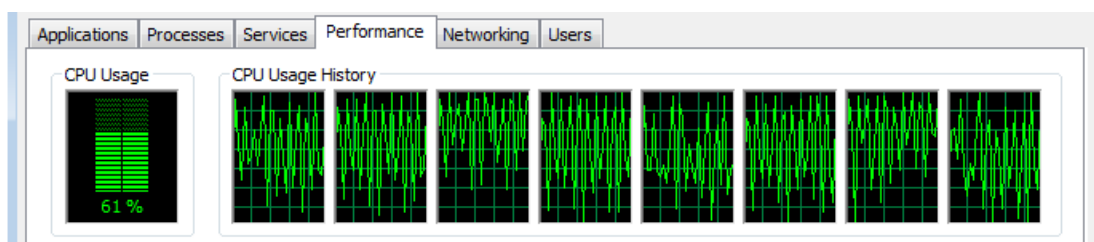
<sup>19</sup>Paralelizace procesů se skládá z rozložení jednoho velkého úkonu do několika menších úkolů, které mohou běžet paralelně. Výsledkem je provedení jednoho úkonu nebo procesu za pomoci více než jednoho procesoru nebo procesorů "Paralelní zpracování", nesmí být zaměňováno se souběžností.

<sup>20</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

## PRAKTICKÁ ČÁST

popis

### **3.1. Návrh**

test

### **3.2. Realizace**

test

## 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] 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/>
- [2] Merta, Zdeněk *Vert.x jOpenSpace 2013* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://jopenspace.cz/2013/presentations/zdenek-merta-vert.x.pdf>
- [3] *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)
- [4] 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/>
- [5] 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/>
- [6] Pitner, Tomáš *Programování v jazyce Java* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://www.fi.muni.cz/~tomp/slides/pbl62/printable.html>
- [7] Lažanský, J. *Procesy a vlákna* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://labe.felk.cvut.cz/vyuka/A4B33OSS/Tema-03-ProcesyVlakna.pdf>
- [8] Fox, Tim *Event loops* [online]. [cit. 2014-03-22]. Dostupný z WWW: <http://vertx.io/manual.html#event-loops>
- [9] Pele, *Diplomka v L<sup>A</sup>T<sub>E</sub>Xu* [online]. [cit. 2008-09-28]. Dostupný z WWW: [pele.gzk.cz/node/37](http://pele.gzk.cz/node/37)
- [10] Roubal, Jiří. *Jirkovy stránky* [online]. [cit. 2008-09-28]. Dostupný z WWW: [dce.felk.cvut.cz/roubal/](http://dce.felk.cvut.cz/roubal/)

- [11] Vydra, Vítězslav. *Počestění LyXu* [online]. 2008 [cit. 2008-09-28]. Dostupný z WWW: [people.fsv.cvut.cz/~vydra/lyxcesky.htm](http://people.fsv.cvut.cz/~vydra/lyxcesky.htm)
- [12] *Písmo T<sub>E</sub>X-Gyre* [online]. [cit. 2008-09-28]. Dostupný z WWW: [www.gust.org.pl/projects/e-foundry/tex-gyre](http://www.gust.org.pl/projects/e-foundry/tex-gyre)
- [13] *Neues Kapitel-Layout* [online]. [cit. 2008-09-28]. Dostupný z WWW: [www.thesis-template.de/archives/5#more-5](http://www.thesis-template.de/archives/5#more-5)
- [14] Vavrečková, Šárka. *Úprava dokumentů* [online]. [cit. 2008-09-28]. Dostupný z WWW: [axpsu.fpf.slu.cz/~vav10ui/obsahy/dipl/typografie.pdf](http://axpsu.fpf.slu.cz/~vav10ui/obsahy/dipl/typografie.pdf)
- [15] *Zdroje informací pro diplomové práce, SLU* [online]. [cit. 2008-09-28]. Dostupný z WWW: [axpsu.fpf.slu.cz/~vav10ui/obsahy/dipl/typodipl.html](http://axpsu.fpf.slu.cz/~vav10ui/obsahy/dipl/typodipl.html)
- [16] *V čem napsat diplomovou práci* [online]. [cit. 2008-09-28]. Dostupný z WWW: [www.student.cvut.cz/cwut/index.php/Diplomová\\_práce#V\\_.C4.8Dem\\_napsat\\_diplomovou\\_pr.C3.A1ci](http://www.student.cvut.cz/cwut/index.php/Diplomová_práce#V_.C4.8Dem_napsat_diplomovou_pr.C3.A1ci)
- [17] Menoušek, Jiří. *Jak (ne)napsat diplomovou a dizertační práci* [online]. [cit. 2008-09-28]. Dostupný z WWW: [www.csmo.cz/other/dizert.php](http://www.csmo.cz/other/dizert.php)
- [18] Polách, Eduard. *Pravidla sazby diplomových prací* [online]. [cit. 2008-09-28]. Dostupný z WWW: [home.pf.jcu.cz/~edpo/pravidla/pravidla.html](http://home.pf.jcu.cz/~edpo/pravidla/pravidla.html)
- [19] Farkašová, Blanka, Krčál, Martin. *Projekt bibliografické citace* [online]. [cit. 2008-09-28]. Dostupný z WWW: [www.citace.com](http://www.citace.com).

Tento seznam literatury byl vytvořen přímo v Lyxu pomocí stylu „Bibliografie“ a generátoru citací [19]. Pořadí citací je takové, jak je sami napíšeme.

Tento seznam literatury byl vytvořen pomocí B<sub>W</sub>T<sub>E</sub>Xu s použitím stylu `csplainnat`. Citace jsou automaticky seřazeny podle abecedy.



# **Přílohy**

(NEÚPLNÝ) PŘEHLED KLÁVESOVÝCH ZKRATEK V L<sub>Y</sub>XU

Klávesová zkratka Alt+p (Paragraph style – výběr stylu odstavce)		
<b>Zkratka</b>	<b>Styl</b>	<b>Mnemo</b>
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		
<b>Zkratka</b>	<b>Činnost</b>	<b>Poznámka</b>
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	