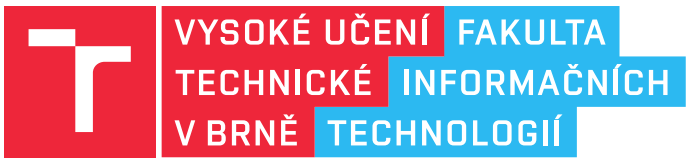
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Modelování a simulace – Počítačové služby (6)

Simulační studie o vytížení jádra serveru počítačové hry Minecraft

6. prosince 2017

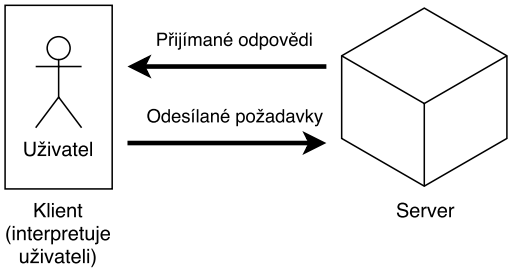
Marek Šipoš (xsipos03)

Obsah

Kapitola 1 Úvod

Optimalizace serverů a zajištění uživatelského komfortu při jejich užívání je velmi frekventovaným a diskutovaným tématem v mnoha moderních službách. K dosažení stabilního (a pokud možno vysokého) výkonu se využívají různé techniky a této oblasti je věnováno často také velké množství prostředků.

Dosažení stability a výkonu je kruciální obzvláště pro služby, kde je vyžadována okamžitá odezva – jedná se například o herní servery. Uživatelé (nebo též *hráči*) typicky pomocí klienta provádějí určité operace, které jsou odesílány na server. Ten je zpracuje a pošle klientovi výsledek jeho konání. Takových operací je často velmi velké množství, patří mezi ně i například pohyb v nějaké hře, kdy klient vysílá neustále informaci o tom, že by se rád přesunul a server pohyb realizuje, načež klient výsledek pohybu okamžitě interpretuje hráči.



Obrázek 1: Ukázka komunikace mezi klientem a serverem

Problém nastává ve chvíli, kdy není realizace okamžitá. Hráči se to typicky jeví tak, že se snaží provést nějakou akci, ale server jeho požadavek nestíhá zpracovat. Klient takový problém řeší různými způsoby – může například akci interpretovat jako vykonanou s tím, že server si to snad „uvědomí později“ (častý případ, obvykle se pak stává, že se akce vrátí zpět, protože server daný požadavek například vyřadil z fronty požadavků a tak k dané akci vlastně nedošlo) nebo klient může na server čekat (v takovém případě čeká i hráč). Oba případy ve výsledku znamenají snížení uživatelského komfortu a je třeba se ptát *proč* k danému zpoždění došlo a jak lze zpoždění v budoucnu předejít.

1.1 Cíl práce

Tato práce si klade za cíl zaměřit se na **situaci, kdy server nestíhá zpracovávat příchozí požadavky**, což vede ke snížení uživatelského komfortu, tuto situaci analyzovat, vyhodnotit a pokusit se o řešení. Zkoumaným serverem je server hry Minecraft. Analýza, vyhodnocení a řešení je realizováno prostřednictvím modelu *systému hromadné obsluhy*[1].

1.2 Postup práce

Aby bylo možno zjistit, proč server nestíhá zpracovávat požadavky, je zjevně nutné mít k takovému serveru přístup (nebo ho přímo vlastnit) a podrobit ho analýzám. Server musí být někde hostován, což může v delším časovém horizontu vyžadovat velké množství prostředků. Navíc nemusí být v určitou chvíli produkční server vůbec k dispozici. Prostředkem analýzy serveru v této práci je pozorování jeho chování za určitých podmínek pomocí vhodných nástrojů, shromáždění dostatku dat, tvorba *validního*[2] *abstraktního*[3] a *simulačního*[4] modelu a následné provedení *simulačních experimentů*[5], na jejichž základě je možné sestavit doporučení a optimalizační techniky.

[1] Systém hromadné obsluhy – IMS (slide č. 136)
[2] Validace modelu – IMS (slide č. 37)
[3] Abstraktní model – IMS (slide č. 42)
[4] Simulační model – IMS (slide č. 44)
[5] Simulace – IMS (slide č. 33)

Vytvořením simulačního modelu odpadá nutnost vlastnit funkční produkční server, takový model je možné poté upravovat a předkládat mu vstupní podmínky, které mohou poukázat na problém ještě předtím, než k němu vůbec nastane (prevence). Vzhledem k charakteristice herních serverů a jejich způsobu zpracovávání požadavků (viz dále) je vhodné server modelovat jako systém hromadné obsluhy.

1.3 Validita modelu

Validita v této práci uváděného modelu byla do určité míry ověřena následujícím postupem:

1. Získání informací o chování serveru analyticky pomocí vhodných nástrojů
2. Tvorba modelu na základě těchto informací
3. Simulace podobných (zjednodušených) podmínek využitím vytvořeného modelu
4. Konfrontace výsledků simulace (chování modelu) s analyticky získanými daty o chování reálného serveru

Pro účely analýzy bylo nutné model vhodným způsobem zjednodušit, aby s ním bylo možno pracovat dostatečně efektivně. Získaný model se pro zadaný cíl práce jeví jako validní.

1.4 Autor práce

Marek Šipoš (xsipos03) – xsipos03@stud.fit.vutbr.cz

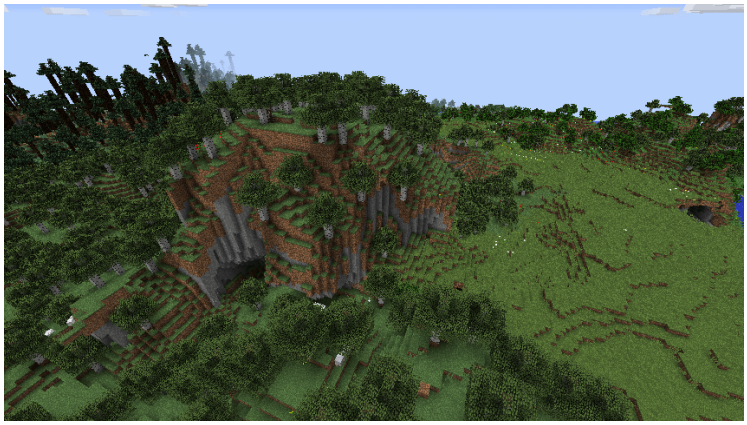
Některé koncepty analýzy Minecraft serveru (zejména týkající se nástroje `Spigot Timings`) byly konzultovány s autory nástroje na diskuzním fóru serverového jádra Spigot[6] a také prostřednictvím serveru IRC.

Kapitola 2 Rozbor tématu a použitých metod/technologií

Předmětem této práce je analýza serverového jádra hry Minecraft. Aby bylo možno o serveru a jeho prvcích (tedy o *systému*[7]) zjistit podstatné informace, je nutné vědět, jak server hry Minecraft vnitřně funguje.

2.1 Minecraft

Minecraft je sandboxová hra, ve které se celý herní svět skládá z bloků. Úkolem hráče je sbírat suroviny, vyrábět herní předměty a přežít proti příšerám, které se v herním světě objevují („spawnují“).



Obrázek 2: Ukázka ze hry Minecraft

[6] Diskuzní fórum Spigot – spigotmc.org/forums
[7] Systém – IMS (slide č. 18)

Hra obsahuje režim pro jednoho i více hráčů. Režim pro více hráčů je realizován více způsoby:

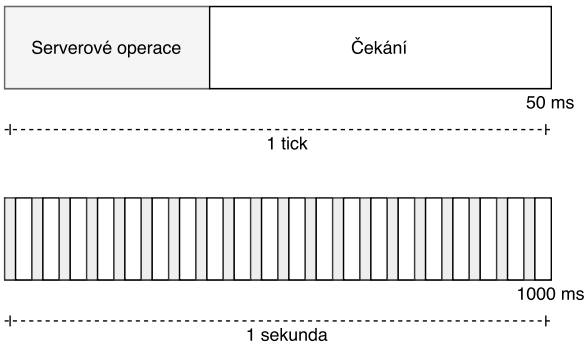
1. Hra po LAN síti – jeden hráč se stane „serverem“ a druhý hráč se na něj napojí. Hra po místní síti.
2. Běžné servery – Mojang (společnost, která hru Minecraft vyvíjí) poskytl hráčům binární soubor se serverem, který umožňuje provoz vlastního Minecraft serveru na veřejné IP. Hráči poté stačí IP adresu zadat a přidat si ji do místního seznamu serverů. Takovým serverům se říká *vanilla* servery (původní, nemodifikované).
3. Realms – jedná se o oficiální hostingovou službu, která umožňuje jednoduchou tvorbu a správu herních serverů. Servery jsou vanilla s možností rozšíření o jednoduché služby poskytované přímo hostingem.
4. Modifikované servery – vzhledem k četné komunitě vývojářů a fanoušků, kteří se hrou Minecraft zabývají, vznikly různé modifikace původního vanilla Minecraft serveru. Tyto servery umožňují vývojářům upravovat jejich chování. Existují různá API, která toto umožňují. Tato práce bude využívat API modifikovaného serveru *Spigot* k získání analytických údajů (viz níže).

2.2 Spigot

Spigot je modifikovaný open-source server hry Minecraft, který umožňuje prostřednictvím tzv. *pluginů*[8] řídit jeho chování. Server jako takový musí řešit obrovské množství událostí – ať už se jedná o spawnování entit, generování a načítání mapy, rozesílání paketů hráčům, řízení chatu, tzv. block-updates[9] a spousta dalších.

2.2.1 tick

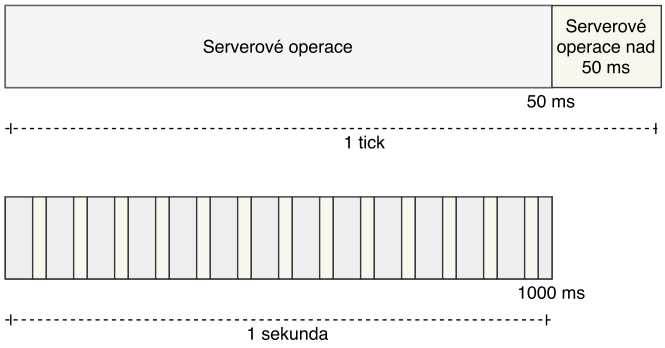
Minecraft server podobně jako mnoho her obsahuje určitou herní smyčku, ve které se vykonávají všechny podstatné události. V Minecraftu se tato herní smyčka skládá z jednotek nazvaných *tick*. V optimálním případě (kdy server vše stíhá) se za 1 sekundu provede 20 ticků (pro tuto veličinu je zavedena zkratka TPS – ticks per second). Každý tick by tedy měl trvat přesně 50 ms a v tomto čase jsou serverem zpracovány všechny čekající události daného ticku.



Obrázek 3: Ideální časové rozložení ticků

Problém nastává ve chvíli, kdy server nestíhá. V takovém případě trvá vykonání událostí delší dobu než 50 ms a tudíž se za 1 sekundu provede méně, než 20 ticků (TPS < 20). Koncoví uživatelé (hráči) tento jev pozorují tak, že se vše zpomalí. Zpomalí se pohyb NPC[10], bloky nebudou aktualizovány jak by měly, herní svět se bude načítat pomaleji, atd. Pokud má server velké problémy udržet tempo, mohou se objevovat problémy zmíněné v kapitole 1, tedy například pokud hráč rozbije blok, blok se může vrátit zpět (a eventuálně se „opět rozbít“, když se k tomu server dopracuje).

[8] Spigot plugin – soubor typu .jar, který se spouští při startu serveru a doplňuje události na serveru o dodatečné chování
[9] Block-updates – interakce s vnitřním stavem bloků, ze kterých se skládá herní svět
[10] NPC – Non-playable character, prakticky jakákoliv aktivní entita jiná než hráč

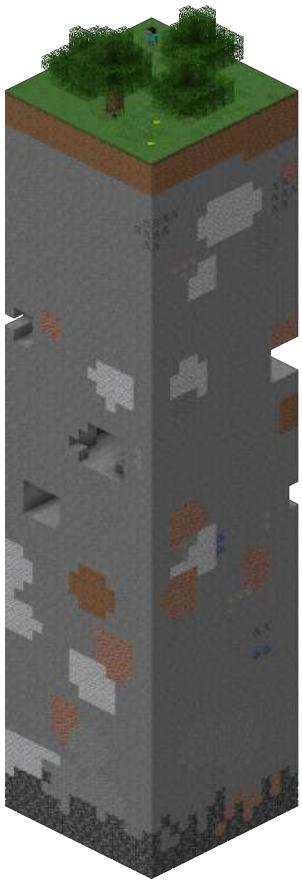


Obrázek 4: Časové rozložení ticků, když server nestíhá

2.2.2 Chunk

Jak již bylo zmíněno výše, celý svět ve hře Minecraft se skládá z bloků. Načítání bloků a rendrování herního světa po blocích by bylo nepraktické a neúčinné. Z tohoto důvodu se bloky seskupují do tzv. Chunků.

Chunk je část mapy skládající se z celkem 65 536 bloků (16x16x256) a na serveru je uložen jako zvláštní soubor ve složce `Region`. Server tyto chunky distribuuje hráčům a zpracovává změny, které v nich nastanou.



Obrázek 5: Chunk (*zdroj: Gamepedia*)

2.3 Použité postupy a prostředky pro sestavení modelu

Protože pro účely této práce nebyl k dispozici plně rozvinutý herní server s velikým množstvím aktivních hráčů, bylo zapotřebí informace o jeho zatížení získat jiným, uměle navozeným způsobem, který se pokud možno co nejvíce přiblíží běžné aktivitě na serveru.

Stav serveru byl monitorován nástrojem *Spigot Timings* a vlastním Java pluginem, který monitoroval průběh TPS v průběhu času a ukládal tento průběh do textového souboru. Tyto dva přístupy zajistily jak informace o spotřebě serverového času obsluhou různých požadavků, tak informace o TPS (udávající hráčský komfort). Různé podmínky během shromažďování informací obstarávaly techniky a příkazy, pomocí kterých byly na server požadavky vhodně zasílány a uměle realizováno běžné chování zatíženého serveru.

Data získaná sledováním výkonu serveru za různých podmínek byla poté použita k vytvoření abstraktního modelu, který byl dále implementován prostřednictvím jazyka C++ a knihovny SIMLIB[11] do simulačního modelu (viz dále). Výsledný model se poté využíval k dalším simulacím, ze kterých byly vyvozeny možné optimalizace.

2.3.1 Fyzický server

Zkoumaný herní server se nacházel na stroji hostovaném společností *Fakaheda.eu* s následujícími parametry: Intel Core i7-5820K, 2 GB RAM, 10 GB SSD, Java SE 8u112, 10 Gigabit optická páteřní linka.

Poznámka: Informace o parametrech mají čistě informační charakter. Cílem této práce je nabídnou možnosti optimalizace pro servery hry Minecraft obecně – bez přímé návaznosti na jakýkoliv parametr fyzického

[11] SIMLIB – Odkaz na stránky

stroje, na kterém server běží. Zkoumat případné problémy při nedostatečných parametrech serveru není předmětem této práce.

2.3.2 Spigot Timings

Součástí funkcionality Spigot serveru je také zabudovaný nástroj *Spigot Timings*, který slouží k monitorování událostí po určité časové období. Nástroj dokáže zjistit, které události trvaly serveru nejdéle a kolik se za určité časové období podařilo zpracovat ticků – tedy o kolik se herní postup za časové období posunul.

Tento nástroj je užitečný zejména v případě, kdy jsou zaznamenány problémy a zpoždění na produkčním serveru s velkým počtem pluginů a/nebo hráčů. V této práci bude nástroj využit k získaní analytických údajů o schopnosti serveru zpracovávat události během určitých situací.

2.4 Původ a vhodnost postupů pro sestavení modelu

K vypracování modelu byly použity tyto prostředky:

- Server Spigot 1.12.2
- Spigot Timings
- TPS meter (vlastní Spigot plugin, Java)
- Vanilla příkazy pro navození podmínek specifických pro tuto práci
- Petriho síť[12] k vymodelování abstraktního modelu
- Knihovna SIMLIB jazyka C++ k vytvoření simulačního modelu

Informace o chování serveru během jednotlivých podmínek byly získány pozorováním, shromažďováním dat, diskuzí na komunitních serverech a studiem zdrojových kódů a dokumentací. Repozitáře serverové nadstavby jsou k dispozici na těchto adresách:

- `https://hub.spigotmc.org/stash/projects/SPIGOT/repos/bukkit/browse`
- `https://hub.spigotmc.org/stash/projects/SPIGOT/repos/craftbukkit/browse`
- `https://hub.spigotmc.org/stash/projects/SPIGOT/repos/spigot/browse`

Bukkit je rozhraním umožňující vytváření pluginů vývojáři. Ve skutečnosti je toto rozhraní i nadále využíváno a Spigot je nadstavbou této „nadstavby“.

CraftBukkit je přímou realizací nad serverovým jádrem původní hry, jedná se o prostředek, díky kterému je vůbec možné pomocí pluginů chování serveru ovlivňovat.

Zdrojový kód původního serveru je z bezpečnostních důvodů obfuskován, proto není jeho studium bez deobfuskace téměř možné. Z toho důvodu není možno sledovat interní procesy Minecraft serveru dopodrobna, z chování lze ovšem vyvodit několik domněnek. Tento princip je uplatněn i v této práci.

Surový protokol hry Minecraft a jeho pakety jsou poměrně slušným způsobem vylíčeny na této stránce: `http://wiki.vg/Protocol`. Díky tomu je možné částečně vydedukovat úkony, které Minecraft server musí provádět, a také složitost těchto operací. Na základě několika vytipovaných paketů jsou poté v kapitole 2.5 prováděna měření.

[12] Petriho síť – IMS (slide č. 123)

2.5 Měření na serveru

Tato podkapitola stručným způsobem popisuje data, která byla na serveru shromážděna prostřednictvím vlastního Java pluginu TPS meter a nástroje Spigot Timings.

Měření byla zaměřena hlavně na práci s chunky (generování, zasílání) a zpracování entit. Server toho samozřejmě dělá MNOHEM více, ovšem:

- K dispozici nebyl server s mnoha hráči
- A nebyl k dispozici ani zavedený server s mnoha pluginy

Tyto dvě položky generují také zatížení, se kterým je často problém. Protože k takovému serveru nebyl při vypracovávání práce přístup, byla měření omezena na výše zmíněné problémové oblasti. Cílem práce je totiž nabídnout případné optimalizační techniky, nikoliv vytvořit dokonalý a všezahrnující model.

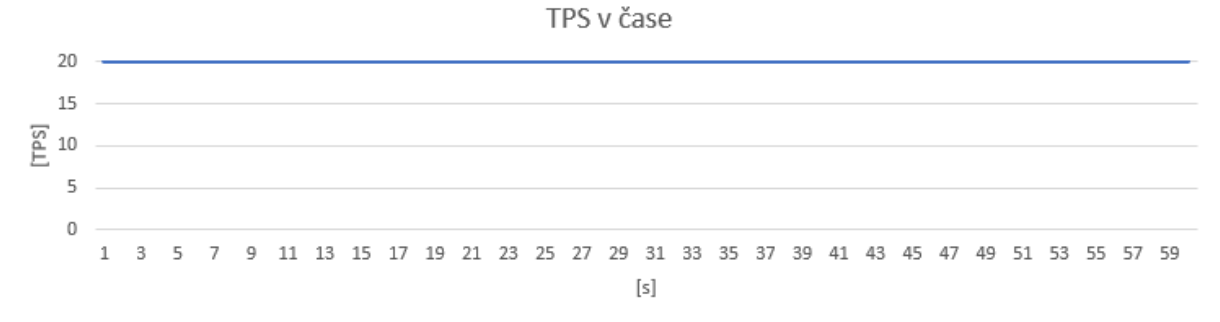
2.5.1 Klidový stav

Předmětem počátečního měření bylo zjistit, jaké chování server vykazuje při minimálním zatížení. Na serveru nebyli žádní hráči a herní mapa byla minimálně načtená (pouze oblast okolo „spawnu“).

Pct Total	Pct Tick	Total	Avg	PerTick	Count	Event
0.76%	0.76%	0.58 s	0.38 ms	1.0	1.5k	Full Server Tick
0.63%	0.63%	0.48 s	0.31 ms	1.0	1.5k	world - entityTick
0.60%	0.60%	0.46 s	0.30 ms	124	188.7k	tickEntity
0.55%	0.55%	0.42 s	0.27 ms	18	27.1k	Activated Entities
0.27%	0.27%	0.20 s	0.13 ms	7.7	11.8k	tickEntity - EntityRabbit
0.25%	0.25%	0.19 s	0.13 ms	18	27.1k	livingEntityAI
0.21%	0.21%	0.16 s	0.10 ms	7.5	11.4k	tickEntity - EntityHorse
0.15%	0.15%	0.12 s	0.08 ms	18	27.1k	livingEntityAIMove
0.13%	0.13%	0.10 s	0.07 ms	18	27.1k	entityMove
0.10%	0.10%	0.08 s	0.05 ms	18	27.1k	livingEntityBaseTick

Obrázek 6: Spigot Timings v klidovém stavu

Z tohoto výpisu je patrné, že server měl největší „práci“ s řízením aktivních entit (typicky zvířata nacházející se v právě načteném kusu mapy). Ostatní úkony byly zcela zanedbatelné.



Obrázek 7: Graf TPS v klidovém stavu

Jak se předpokládalo, zatížení je tak malé, že server bez sebemenších problémů všechny úkony zvládá včas. TPS je po celou dobu stabilně 20.

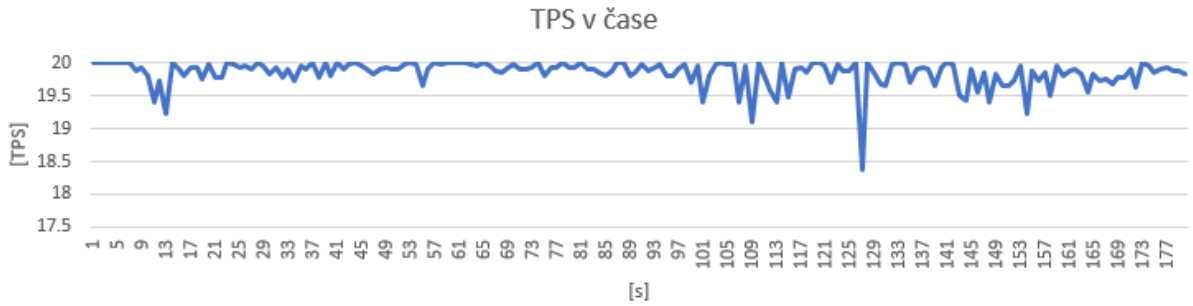
2.5.2 Pohyb po nevygenerované mapě

Předmětem dalšího měření byl pohyb po nevygenerované mapě s jedním aktivním hráčem. Server v tomto případě musel načítat chunky okolo tohoto hráče poprvé, což znamená generovat herní svět za běhu.

Pct Total	Pct Tick	Total	Avg	PerTick	Count	Event
23.25%	23.43%	42.32 s	11.71 ms	1.0	3.6k	Full Server Tick
18.77%	18.91%	34.17 s	9.46 ms	1.0	3.6k	world - doChunkMap
18.27%	18.14%	33.26 s	9.07 ms	1.2	4.4k	world - syncChunkLoad
10.13%	10.20%	18.43 s	5.10 ms	1.3	4.7k	world - chunkLoad - Post
1.59%	1.60%	2.90 s	0.80 ms	1.0	3.6k	world - entityTick
1.54%	1.55%	2.81 s	0.78 ms	228	823.2k	tickEntity
1.48%	1.49%	2.69 s	0.74 ms	52	186.4k	Activated Entities
1.31%	1.32%	2.39 s	0.66 ms	1.0	3.6k	world - doTickTiles
0.71%	0.71%	1.28 s	0.36 ms	39	141.9k	livingEntityAI
0.40%	0.41%	0.74 s	0.20 ms	1.0	3.6k	world - doTickPending

Obrázek 8: Spigot Timings při načítání nových chunků

Nástroj Spigot Timings zde ukazuje zajímavou informaci – ze 180 sekund měřeného času server 34 sekund strávil jen operacemi týkajícími se mapy. Z toho vyplývá, že generování nových chunků mapy je opravdu náročná operace.



Obrázek 9: Graf TPS při načítání nových chunků

Jak lze v grafu vyčíst, server mírně nestíhal, což způsobilo pokles TPS. Stabilních 20 TPS prvních pár sekund bylo způsobeno tím, že mapa byla okolo hráče již načtená a chvíli trvalo, než se hráč dostal na její okraj, aby se musela načíst (a vygenerovat) nová.

2.5.3 Pohyb po vygenerované mapě

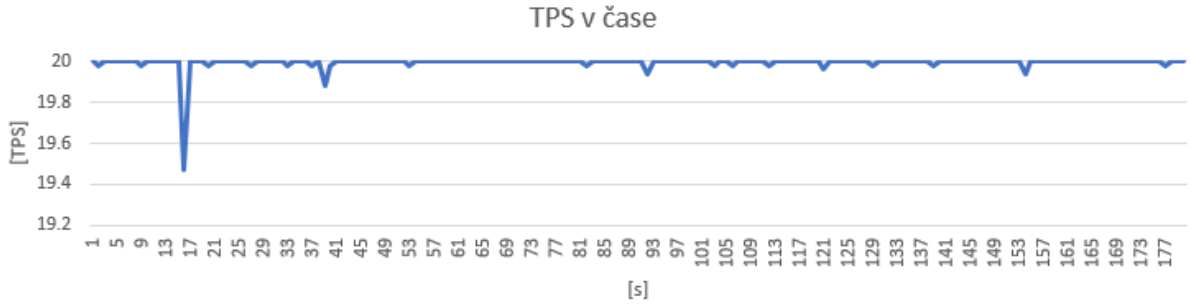
Může být zajímavé pozorovat, jaký rozdíl způsobí, když bude mapa již vygenerovaná. Pomocí pluginu *WorldBorder*[13]byla mapa nejdříve předgenerována v poloměru 1 000 bloků a až poté se hráč v tomto poloměru po mapě začal pohybovat.

[13]WorldBorder – https://dev.bukkit.org/projects/worldborder

Pct Total	Pct Tick	Total	Avg	PerTick	Count	Event
2.96%	2.96%	5.44 s	1.48 ms	1.0	3.7k	Full Server Tick
1.11%	1.11%	2.05 s	0.56 ms	1.0	3.7k	world - entityTick
1.05%	1.05%	1.94 s	0.53 ms	267	983.1k	tickEntity
0.97%	0.97%	1.79 s	0.49 ms	36	133.7k	Activated Entities
0.44%	0.44%	0.81 s	0.22 ms	15	55.2k	tickEntity - EntityRabbit
0.39%	0.39%	0.72 s	0.20 ms	1.0	3.7k	world - doTickTiles
0.38%	0.38%	0.70 s	0.19 ms	33	122.1k	livingEntityAI
0.36%	0.37%	0.66 s	0.18 ms	1.0	3.6k	world - chunkLoad - Structures
0.31%	0.31%	0.56 s	0.15 ms	1.0	3.7k	world - mobSpawn
0.30%	0.30%	0.55 s	0.15 ms	37	137.2k	entityMove

Obrázek 10: Spigot Timings při načítání již vygenerovaných chunků

Ukazuje se, že když server nemusí mapu generovat, je mnohem méně zatížen. Možná překvapivé zjištění je, že server měl více práce se samotnými entitami, než s odesíláním již vygenerovaných Chunků hráči. Je to tím, že Chunk se zkrátka odešle, jedna operace, kdežto entity vytvářejí požadavky neustále.



Obrázek 11: Graf TPS při načítání již vygenerovaných chunků

TPS bylo až na malé výkyvy stabilní. Okolo šestnácté vteřiny došlo k menšímu výkyvu, tyhle „lag spikes“ se často vyskytují, když Garbage Collector Javy uvolňuje paměť.

2.5.4 Spawnovaní entit

Na reálném serveru se entity spawnují prakticky neustále. Hráči prozkoumávají mapu, server spawnuje nové a nové entity. Hráči dále mohou rozmnožovat domestikovaná zvířata nebo stát u tzv. „spawneru“, který v určitém intervalu vytváří entity nové.

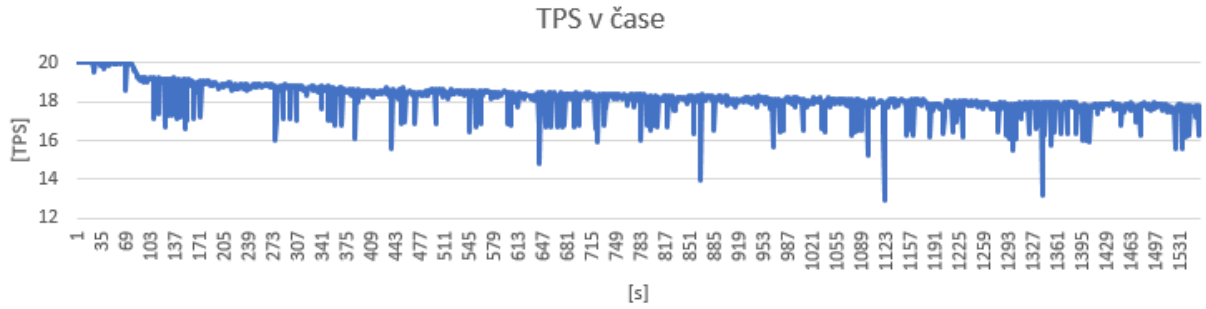
Cílem tohoto měření bude znázornit vliv počtu entit na výkon serveru. Za tímto účelem byl vytvořen jednoduchý redstone[14] obvod, na který byl připojen command block[15] s vhodně zapsaným `/summon` příkazem, který entity spawnuje. Obvod je aktivován opakovaně a tím se postupně počet entit neustále zvyšuje. Každý tick je spawnuta jedna entita.

[14]Redstone – sada speciálních herních předmětů, které umožňují tvořit jednoduché logické obvody podobné těm elektronickým
[15]Command block – speciální herní blok, který po aktivaci redstone signálem spustí zadaný příkaz

Pct Total	Pct Tick	Total	Avg	PerTick	Count	Event
96.95%	106.50%	1,659.70 s	53.25 ms	1.0	31.2k	Full Server Tick
87.60%	96.22%	1,499.60 s	48.11 ms	1.0	31.2k	world - entityTick
87.32%	95.91%	1,494.81 s	47.96 ms	1,372	42,759.4k	tickEntity
86.89%	95.44%	1,487.45 s	47.72 ms	829	25,837.1k	Activated Entities
85.53%	93.95%	1,464.20 s	46.97 ms	806	25,120.6k	tickEntity - EntityPig
42.64%	46.83%	729.93 s	23.42 ms	829	25,838.0k	entityMove
42.60%	46.80%	729.32 s	23.40 ms	812	25,293.8k	livingEntityAIMove
30.44%	33.44%	521.10 s	16.72 ms	812	25,293.8k	livingEntityAICollision
8.43%	9.26%	144.30 s	4.63 ms	812	25,293.8k	livingEntityAI
3.42%	3.75%	58.47 s	1.88 ms	812	25,293.8k	livingEntityBaseTick

Obrázek 12: Spigot Timings během spawnování entit

Měření trvalo celkem 1667 sekund. Procesor tento čas téměř zcela využil. Minecraft a speciálně Spigot obsahuje vnitřní mechanismy, které zabraňují úplnému přetížení serveru entitami – v tomto případě byl nejspíše využit mechanismu *nerfed entities*, který způsobí, že se chování entit upraví tak, aby nezasílalo tolik požadavků na server. To způsobí ještě větší snížení hráčského komfortu z hlediska interakce s těmito entitami, ale server tuto situaci „rozchodí“.



Obrázek 13: TPS během spawnování entit

Z obrázku je vidět, že první minutu a pár sekund server entity zvládal a hodnota TPS zůstala relativně stabilně na 20. Jakmile server přestal stíhat, TPS šla strmě dolů. Server tuto situaci vyhodnotil správně a aktivoval režim *nerfed entities*, čímž pokles TPS úspěšně zpomalil. Na konci měření byla TPS okolo 18.

Dále je možné všimnout si pravidelných „lag spikes“ způsobených Garbage Collectorem. Toto snížení výkonu nelze zachytit pomocí nástroje Spigot Timings, protože nejsou měřitelné v rámci hry. Jedná se o snížení výkonu nižší vrstvy (blíže CPU) související s režií jazyka Java.

Poznámka: Režim Nerfed entities způsobuje pokles množství požadavků o cca 90 %, ale tyto požadavky (resp. jejich zpracování) trvá o něco déle. Je to daň za režii spojenou s tímto režimem.

Kapitola 3 Koncepce

Pro vyvození optimalizačních technik bylo využito Petriho sítě namodelované pomocí knihovny SIMLIB v jazyce C++. Systém (server) byl modelován jako systém hromadné obsluhy (dle zadání). K vytvoření modelu byla vhodně využita fakta z předchozí kapitoly.

3.1 Vhodnost vyjádření pomocí SHO

Využití SHO k vyjádření procesů systému herního serveru je vhodné z hlediska zpracovávání požadavků. Jakýkoliv úkon, který server musí provést, si lze představit jako požadavek, který je na server zaslán. Požadavky

jsou zasílány s různým časovým odstupem, stochasticky nebo i v pravidelném intervalu. Server některé typy požadavků vyřizuje přednostně, ostatní ukládá do fronty. Podle množství požadavků se může přepnout režim jejich zpracovávání a parametry linek. To vše lze pomocí SHO modelovat a odpovídá to tomu, jak server pracuje.

3.2 Převod faktů do modelu

V průběhu ticku přicházejí na server požadavky. Jejich zpracování může trvat různou dobu a mohou přicházet s různou pravidelností. Server může požadavky zpracovávat v různých režimech a také i mimo hlavní herní vlákno (např. chat nebo požadavky pluginů). Zpracování požadavků vyžaduje určitý serverový čas, který, pokud je příliš veliký, snižuje TPS. V této podkapitole jsou tyto záležitosti shrnuty tak, aby je bylo možno pojmout zjednodušeným modelem.

3.2.1 Požadavky na server

U každého požadavku je důležité mít přehled, jak dlouho přibližně trvá jeho zpracování. Tento údaj se dá vyčíst ze Spigot Timings a je specifický vždy pro daný server. Například:

Pct Tick	Total	Avg	PerTick	Count	Event
3.17%	2.53 s	1.59 ms	1.0	1.6k	Full Server Tick
1.03%	0.82 s	0.51 ms	1.0	1.6k	world - entityTick
0.97%	0.77 s	0.49 ms	237	378.3k	tickEntity

Obrázek 14: Část výpisu nástroje Spigot Timings

Chceme zjistit, jak dlouho trvá jedna operace `tickEntity` (zpracování entity během ticku).

Pct Tick	Kolik procent doby ticku zpracování událostí tohoto typu zabralo
Total	Serverový čas strávený zpracováním událostí tohoto typu
Avg	Průměrná doba, kterou události tohoto typu v 1 ticku zabraly
PerTick	Kolik událostí tohoto typu bylo v 1 ticku průměrně zpracováno
Count	Celkový počet zpracovaných událostí tohoto typu

Doba trvání jedné události se tedy dá snadno vypočítat tímto způsobem:

$$t = \frac{\text{Avg}}{\text{PerTick}}$$

V tomto případě zpracování jedné události typu `tickEntity` trvalo přibližně $206,75\,\mu m$ serverového času.

Pro účely modelu budou použity následující požadavky s dobami zpracování a generátory[16]:

#	Požadavek	Doba zpracování	Generování
1	Zpracování entity (běžný režim)	0,002 ms	EXP(0,2 ms)
2	Zpracování entity (nerfed entities)	0,035 ms	EXP(1,8 ms)
3	Generování mapy	7 ms + (3 ms * players)	max 1x za tick, players > 0
4	Načtení mapy	0,02 ms + (0,01 ms * players)	max 1x za tick, players > 0
5	Požadavek pluginu	0,005 ms * players	EXP(1 ms)
6	Garbage Collector	500 ms	Každých 200 sekund

[16] Generátor – IMS (slide č. 97)

- Zpracování požadavků entit v běžném režimu je rychlejší, ale v režimu Nerfed Entities je těchto požadavků až o 90 % méně.
- Generování mapy je zdaleka nesložitější požadavek, nicméně reálně se zas tak často nevykonává, hráči mají tendenci hrát v určitém okolí (mapě), které znají.
- Generování a načítání mapy se vykonává v ticku maximálně jednou. Po zpracování požadavku v daném ticku jsou všechny chunky pro daný tick zpracovány. To je důvod, proč je doba zpracování závislá na počtu hráčů.
- Pro jednoduchost byla operace načtení mapy sloučena s jejím uvolněním, které se chová podobně.
- Požadavky pluginu představují veškeré operace těchto pluginů. Tato část je velmi diskutabilní a specifická pro daný server a pluginy samotné. Plugin jako takový může dělat takřka cokoliv a záleží na tom, jak je naprogramován. Čas a generování pluginu v této práci vychází z váženého průměru Spigot Timings, které byly náhodně nalezeny na diskuzním fóru Spigot komunity. Pluginy mohou využívat i dalších vláken mimo hlavní serverové (maximální počet vláken představuje parametr *max threads*), takže je možné zpracovávat více požadavků pluginů najednou (sklad[17]).
- Aby bylo možno pomocí SHO zaznamenat snížení výkonu způsobeného Garbage Collectorem, bude pro jednoduchost Garbage Collector uveden jako požadavek. Jeho doba zpracování a generování byla určena postupnou kalibrací modelu.
- Počet hráčů je pro zjednodušení zaveden jako parametr *players*. Počet entit je pro jednoduchost zanedbán a požadavky jsou generovány s exponenciálním rozložením[18].

3.2.2 Kdy server nestíhá?

Nejdůležitější otázkou je, jak pomocí modelu zjistit, že server nebude stíhat. Víme, že jeden tick by měl trvat přesně 50 ms. Pokud trvá déle, server přestává stíhat. Konečná hodnota TPS se určí naskládáním jednotlivých ticků za sebe, dokud všechny dohromady nepřekročí čas 1 s. Pak už jen stačí ticky spočítat.

Předmětem jedné simulace bude jeden tick. Během ticku (který může trvat 50 ms nebo déle) přicházejí na server požadavky. Server tyto požadavky zpracovává a přesně ten čas zpracování je důležitý. Pokud bude spotřebovaný serverový čas 50 ms nebo více, server nestíhá. Čas, kdy server čeká, se do tohoto času nezapočítává. Simulace se budou spouštět alespoň tak dlouho, dokud součet jejich dob trvání nepřekročí 1 000 ms.

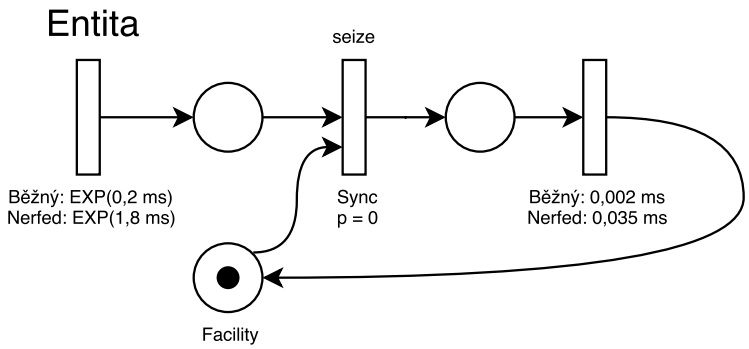
[17] Sklad – IMS (slide č. 146)
[18] Exponenciální rozložení – IMS (slide č. 91)

3.3 Petriho síť

Následuje zjednodušené vyobrazení Petriho sítí, které odpovídají jednotlivým požadavkům. Model představuje simulaci o délce 1 tick. Model obsahuje dvě obslužné linky – *Sync* a *Async* – pomocí kterých server požadavky zpracovává.

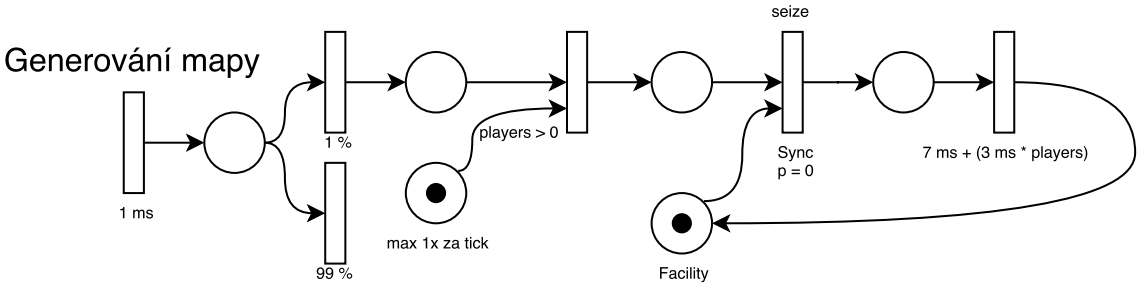
Petriho síť byla pro přehlednost rozdělena, všechny požadavky interagují s tytéž obslužnými linkami.

3.3.1 Zpracování entity



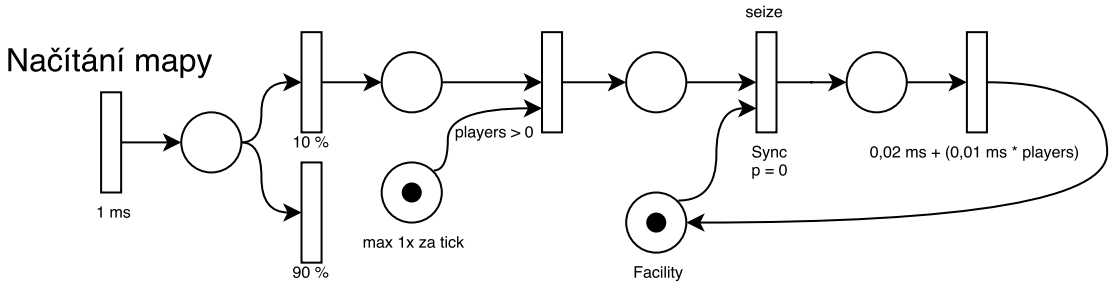
Obrázek 15: Petriho síť – Entita

3.3.2 Generování mapy



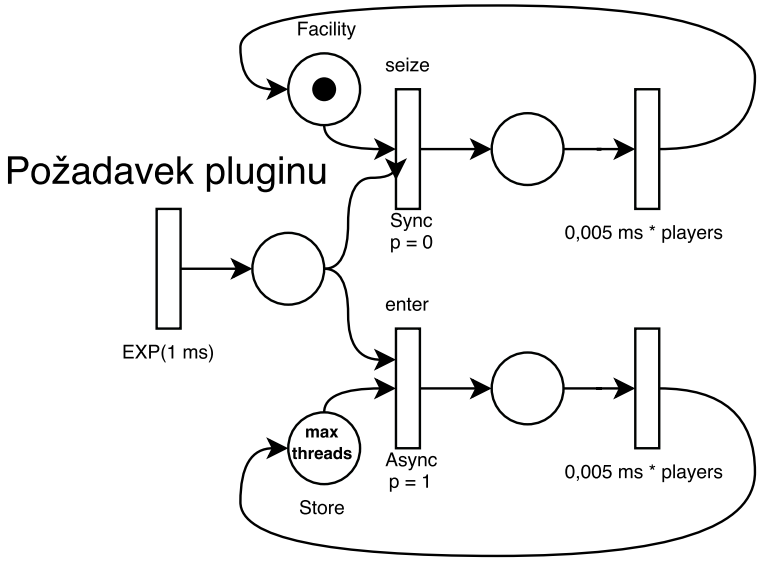
Obrázek 16: Petriho síť – Generování mapy

3.3.3 Načítání mapy



Obrázek 17: Petriho síť – Načítání mapy

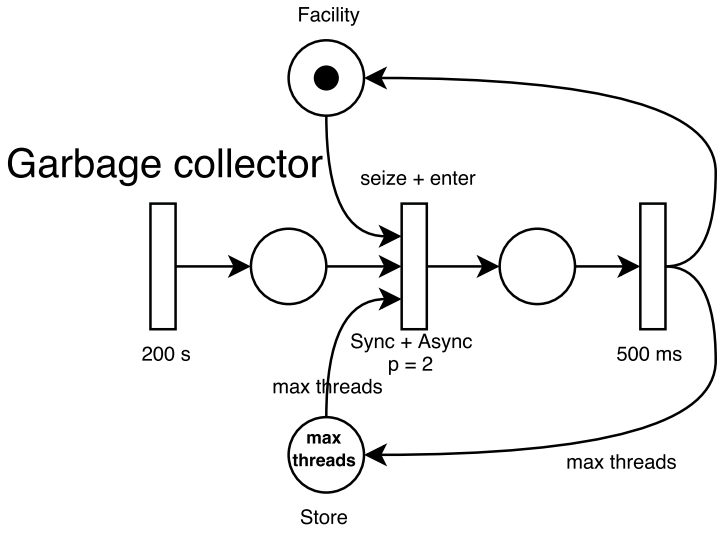
3.3.4 Požadavek pluginu



Obrázek 18: Petriho síť – Požadavek pluginu

Petriho síť předpokládá, že jsou pluginy napsány tak dobře, že vždy upřednostňují asynchronní vlákno, aby neblokovaly synchronní, na kterém probíhají herní operace.

3.3.5 Garbage Collector



Obrázek 19: Petriho síť – Garbage collector

Činnost Garbage Collectoru je reprezentována jako požadavek, který zablokuje veškerou činnost serveru po uvedený čas.

Kapitola 4 Architektura simulačního modelu/simulátoru

Simulační model byl implementován v jazyce C++ za použití knihovny SIMLIB. Tato kapitola objasňuje funkcionalitu a kód simulátoru.

4.1 Obslužné linky

V simulačním modelu byly použity dvě obslužné linky:

1. Sync – zařízení, pro prakticky všechny požadavky
2. Async – sklad s kapacitou *Max threads*, představuje vlákna pro zpracování požadavků pluginů.

Obslužné linky reprezentují server, který zpracovává příchozí požadavky. Zařízení *Sync* pak má přímý vliv na hodnotu TPS a tudíž hráčský komfort. Serverové jádro hry Minecraft nedokáže své interní procesy dělit mezi další vlákna nebo jádra (s výjimkou chatu). Ostatní vlákna jsou využívána pluginy, například pro spojení s databází a pro další blokující operace.

4.2 Princip simulátoru

Program je zahájen vstupním bodem (funkce `main`). Nejdříve se stanoví rozsah modelového času[19] simulátoru. Pro každou simulaci byl použit rozsah 0-1000 s. Během probíhající simulace se generují požadavky, které vstupují a opouštějí jednu z obslužných linek (*Sync* nebo *Async*).

Parametry simulačního modelu jsou reprezentovány globálními konstantami `PLAYERS` a `MAX_THREADS` (počet hráčů na serveru, resp. maximální počet asynchronních vláken).

Režim *nerfed entities* je reprezentován globální proměnnou nerfed_entities. Zapíná se automaticky, kdykoliv TPS klesne pod hodnotu 19.0.

Globální proměnné `genmap_tick` a `loadmap_tick` obsahují informaci, zda již v daném ticku byl vygenerován požadavek pro vygenerování resp. načtení mapy.

Ačkoliv simulace probíhá nepřerušovaně, dochází při ní k dělení na menší časové úseky, pro které se „měří“ spotřebovaný čas CPU. Pro jednoduchost není měření po tickách, nýbrž po celých sekundách (což odpovídá měření v pluginu *TPS meter*). Pro každou sekundu se poté vypočítá TPS

4.3 Použité třídy

Následuje seznam použitých tříd a jejich stručný popis.

4.3.1 Požadavky

Každý požadavek je reprezentován svou vlastní třídou, která popisuje jeho chování pomocí zděděné metody `Behavior()`. Požadavky jsou vytvářeny generátorem (viz podkapitola 4.3.2).

Název	Nadřazená	Účel
Pozadavek_entita	Process	Požadavek na zpracování entity (normální i nerfed entities režim)
Pozadavek_genmap	Process	Požadavek na vygenerování mapy
Pozadavek_loadmap	Process	Požadavek na načtení mapy
Pozadavek_plugin	Process	Požadavek z pluginu (Sync i Async)
Pozadavek_gc	Process	Aktivita Garbage Collectoru

[19] Modelový čas – IMS (slide č. 21)

4.3.2 Generátory

Generátory jsou modelovány jako události, které se před spuštěním simulace aktivují metodou `Activate()` a poté sebe sama naplánují pro budoucí spuštění. Aby na začátku nedošlo k vygenerování všech požadavků na server najednou, dochází ke spuštění se zpožděním – jinak řečeno, první vygenerování v čase $Time = 0$ je vynecháno.

Název	**Nadřazená**
Generator_entita	Event
Generator_genmap	Event
Generator_loadmap	Event
Generator_plugin	Event
Generator_gc	Event

Názvy tříd generátorů jsou odvozeny od názvů tříd příslušných požadavků.

Kapitola 5 Podstata simulačních experimentů a jejich průběh

Cílem simulačních experimentů je prozkoumat chování serveru bez přístupu k produkčnímu serveru a vyvodit případná optimalizační doporučení.

5.1 Plán a podstata

Vzhledem k povaze zkoumaných dat, které byly do simulačního modelu zakomponovány, bylo provedeno zkoumání serveru za těchto podmínek:

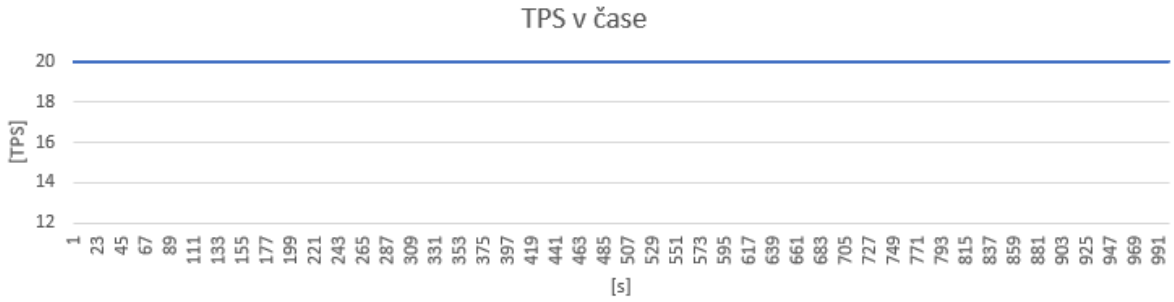
- Běžný provoz (10 hráčů)
- Soustavné generování mapy
- Chování serveru bez využití asynchronních vláken
- Provoz velkého serveru (100 hráčů)
- Provoz velkého serveru bez Nerfed entities

Jednotlivé experimenty si kladou za cíl zjistit, jak se server během těchto podmínek chová, zda to odpovídá předpokládanému chování a jak by šlo server optimalizovat.

5.2 Průběh experimentů a optimalizační rady

Simulační model byl spuštěn celkem 5x s různými parametry.

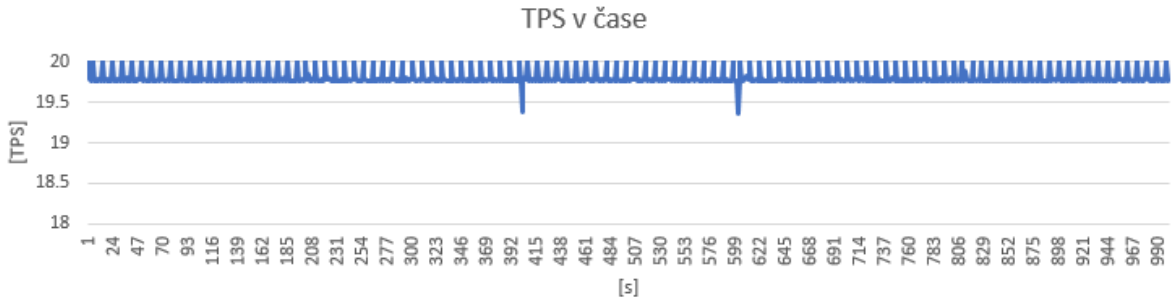
5.2.1 Běžný provoz (10 hráčů)



Obrázek 20: Simulační model – TPS při běžném provozu

Jak se očekávalo, TPS se drží stabilně na 20.0. Žádné překvapení.

5.2.2 Soustavné generování mapy

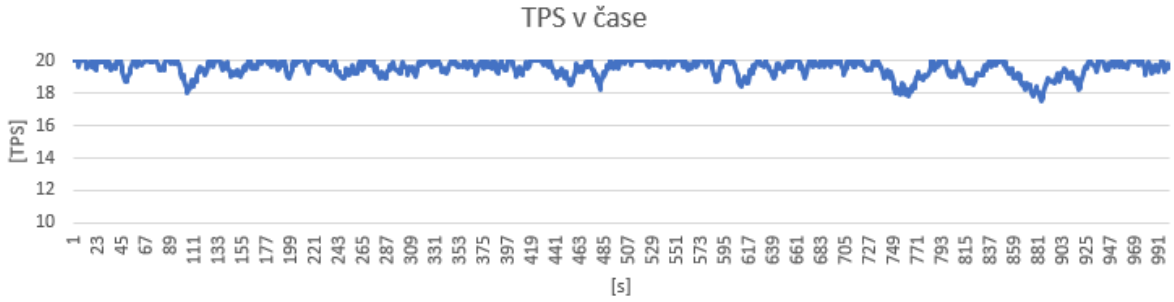


Obrázek 21: Simulační model – TPS při generování mapy

Při soustavném generování mapy měl server již mírné problémy. Víceméně to odpovídá analyticky naměřeným hodnotám. Toto bylo simulováno s 10 hráči. Pokud by probíhalo generování s více hráči, server by již měl větší problémy. DOPORUČENÍ: předvygenerovat si mapu, např. pomocí pluginu *WorldBorder* a pokud možno když na serveru nejsou hráči.

„Zoubky“ v grafu byly způsobeny nejspíše přílišným zjednodušením modelu. Okolo 400 a 600 sekundy lze vidět mírný pokles v důsledku GC.

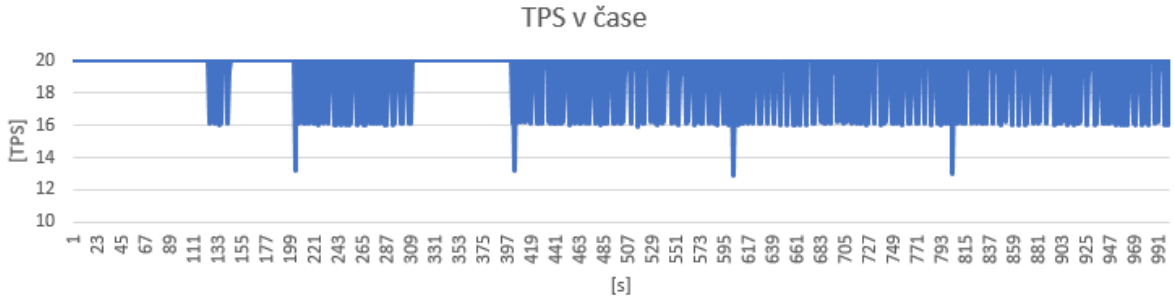
5.2.3 Chování serveru bez využití asynchronních vláken



Obrázek 22: Simulační model – TPS bez využití Async

Jak lze vidět, pluginy mají důrazný vliv na TPS, pokud nejsou spouštěny na Async vlákně, proto je důležité používat OVĚŘENÉ a dobře naprogramované pluginy.

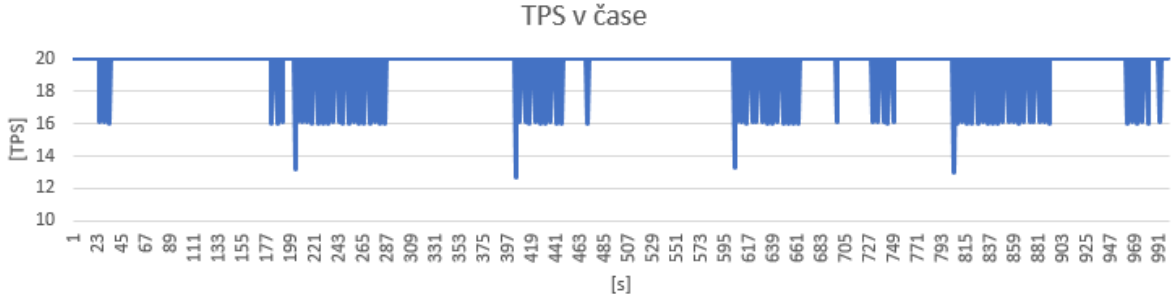
5.2.4 Provoz velkého serveru (100 hráčů)



Obrázek 23: Simulační model – TPS u velkého serveru

Výsledné hodnoty nejsou evidentně validní. Na grafu lze ale vidět práce Garbage collectoru.

5.2.5 Provoz velkého serveru bez Nerfed entities



Obrázek 24: Simulační model – TPS u velkého serveru bez Nerfed entities

Výsledné hodnoty nejsou evidentně validní. Na grafu lze ale vidět práce Garbage collectoru.

Kapitola 6 Shrnutí simulačních experimentů a závěr

Cílem této práce bylo získat data o zpracování požadavků serverovým jádrem Spigot, analyzovat získaná data, vybrat nejdůležitější a nejnáročnější operace a ty poté převést do modelu. Ten bylo nutno naprogramovat a provést s ním simulace takové, aby bylo možno navrhnout optimalizační techniky.

Provedená měření potvrdila očekávání – mezi nejnáročnější operace patří ty, které souvisí s řízením určité entity (jedná se o konstantní proud požadavků), nebo jsou výpočetně náročné (generování mapy).

Ukázalo se, že serverové jádro Spigot se dokáže s přetížením poměrně slušně vypořádat, protože obsahuje vnitřní mechanismy, které upravují chování serveru tak, aby server stihl požadavky zpracovat (např. režim *Nerfed entities*). Tyto změny ovšem přímo ovlivňují hráčský komfort.

Vytvořený simulační model nenapodobuje práci serveru za reálných podmínek dostatečně věrně. Je to způsobeno nejspíše přílišným zjednodušením abstraktního modelu, nedostatkem dat při měření a také absenci produkčního serveru.

Velkou překážkou je nedostatek zdrojů popisujících interní chování serverového jádra a to, že kód serveru je obfuskován, tudíž velmi špatně čitelný. Proto bylo nutno získat data pozorováním.

Navzdory výše uvedenému problému se podařilo vyvodit základní poznatky o chování serveru a stanovit pár základních rad, které by měly při provozu serveru pomoci hráčský komfort zvýšit.

6.1 Možná vylepšení

Získaný model by šel dále vylepšit o zapracování dalších požadavků na server a podrobnějšími měřeními v produkčním serveru s daleko větším vzorkem. Jistě vhodné by bylo také provést měření v závislosti na konfiguračních nastaveních Spigotu. Zajímavé by bylo také porovnat jednotlivá serverová jádra (např. Sponge vs Spigot) z hlediska výkonu, případně provést měření serveru s různými nasazenými veřejně dostupnými pluginy.