Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Síťový plánovač a exekutor dávkových úloh

Bakalářská práce

2023 Martin Joukl

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin Joukl**
Osobní číslo:	**I20107**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Síťový plánovač a exekutor dávkových úloh**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je vytvořit nástroj(e) pro plánování (FIFO, prioritní plánování), automatické provádění úloh na výpočetních uzlech a sběr jejich výsledků. Architektura systému je typu klient-server, kdy server v zásadě řídí jednotlivé výpočetní uzly, zadává jim dílčí úlohy ke zpracování a přijímá výsledky.
V teoretické části bude provedena rešerše obdobných nástrojů pro plánování úloh a jejich provádění v prostředí počítačových clusterů. Dále bude provedena analýza a návrh vlastního systému, resp. jednotlivých aplikací. Server přijímá úlohy od uživatele a reaguje na požadavky/data od jednotlivých výpočetních uzlů. Klienti (výpočetní uzly) se aktivně dotazují (pooling) serveru na nové úlohy, provádějí je a po jejich skončení předávají výsledky serveru. Klienti od serveru při zadání úlohy obdrží kompletní aplikaci ke spuštění a informace, jak má být aplikace spuštěna a kde se budou nacházet výsledky.
Server bude naslouchat na jediném (příchozím) portu, klienti vytvářejí pouze odchozí síťové spojení. Systém by měl korektně řešit problémy důvěrnosti a integrity dat. Systém musí být multiplatformní a podporovat minimálně OS Windows a Linux.

Rozsah pracovní zprávy: **40 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

RUSTY HAROLD, Eliotte. *Java Network Programming*. 3rd ed. USA: O'Reilly Media, 2004. ISBN 9780596552589.

L. CALVERT, Kenneth a Michael J. DONAHOO. *TCP/IP Sockets in Java: Practical Guide for Programmers*. 2nd ed. USA: Morgan Kaufmann, 2011. ISBN 9780080568782.

HOOK, David. *Beginning Cryptography with Java*. John Wiley, 2005. ISBN 9780471757016.

Vedoucí bakalářské práce: **Ing. Roman Diviš, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem Síťový plánovač a exekutor dávkových úloh jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 02. 04. 2023

Martin Joukl

PODĚKOVÁNÍ

Chtěl bych poděkovat panu Ing. Romanovi Divišovi, Ph.D za veškerý strávený čas, který strávil vedením mé práce, poskytováním pomoci, rad a následným revidováním této práce, které mi značně pomohly s jejím vypracováním. Dále bych chtěl poděkovat rodině za poskytnutou oporu během studia.

ANOTACE

Cílem bakalářské práce je návrh a implementace systému pro plánování dávkových úloh v prostředí počítačových clusterů. Implementovaný systém se skládá ze dvou částí, ze serveru a z několika klientů, a umožňuje zadání dávkové úlohy serveru a poté její následné naplánování a vykonání na výpočetních uzlech. Veškerá komunikace mezi serverem a klienty je šifrovaná pomocí asymetrické a následně symetrické kryptografie. Server je aplikace s grafickým rozhraním, klient je konzolová aplikace. Obě aplikace jsou multiplatformní a podporují operační systémy Windows a Linux.

KLÍČOVÁ SLOVA

dávková úloha, plánovač dávkových úloh, Slurm, PBS Professional

TITLE

Network planner and executor of batch jobs

ANNOTATION

The aim of the bachelor's thesis is to design and implement a system for scheduling batch jobs in computer clusters environment. The implemented system consists of two parts: a server and several clients, and allows the submission of batch jobs to the server, followed by their scheduling and execution on computing nodes. All communication between the server and clients is encrypted using asymmetric and afterward symmetric cryptography. The server is an application with graphical user interface, while the client is a console application. Both applications are multi-platform and support Windows and Linux operating systems.

KEYWORDS

batch job, batch job planner, Slurm, PBS Professional

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK

AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
FCFS	First Come First Served
FIFO	First In, First Out
GUI	Graphic User Interface
HPC	High-performance computing
ID	Identifier
MoM	Job management daemon
QOS	Quality of service
REST	Representational State Transfer
RSA	Rivest-Shamir-Adleman
SPF	Shortest Processing time First
TCP	Transmission Control Protocol
TPP	TCP-based Packet Protocol

ÚVOD

Již od počátku výpočetní techniky byl kladen požadavek na vysoký výkon výpočetních počítačových systémů, který je nutný pro provádění výpočetně složitých operací. Tento požadavek trvá i dodnes, a to v oborech napříč nejrůznějšími odvětvími – ať už se jedná o výzkum nebo business [19]. Z tohoto důvodu existují systémy HPC, které tento požadavek efektivně řeší.

V minulosti byly jako systémy HPC chápány monolitické superpočítače, avšak od tohoto konceptu se povětšinou ustoupilo – navyšování jejich výkonu se ukázalo jako problematické, zejména díky problematickému navyšování výkonu jednotlivých procesorů. V dnešní době je pod tímto pojmem chápán distribuovaný systém složený z počítačových clusterů – tedy z většího množství počítačových uzlů (až v řádech desetitisíců uzlů), které společně tvoří jeden systém [20]. Tento sytém poskytuje požadovaný výpočetní výkon a umožňuje snadné škálování, avšak jeho provoz je velmi nákladný – proto musíme řešit jeho efektivní využití.

Existuje mnoho způsobu, jak systém efektivně využít – jedním z nejvíce rozšířených způsobů běhu úloh na těchto systémech je spouštění dávkových úloh. Zde největším úskalím je efektivní plánování těchto úloh. Tento problém řeší plánovače úloh, na které se dále zaměříme.

V první části této práce představíme důležité pojmy týkající se běhu dávkových úloh v prostředí počítačových clusterů a dále provedeme rešerši nejznámějších veřejně dostupných plánovačů – konkrétně shrneme a porovnáme systémy *PBS Professional*® a *Slurm Workload Manager*, které slouží k plánování dávkových úloh. Následně systémy dále porovnáme s vlastním implementovaným řešením.

V druhé části práce představíme návrh a implementaci vlastního systému pro plánování dávkových úloh. Tento systém je čerpá inspiraci z předešlých systémů, avšak v porovnání s předešlými je zaměřen na jednodušší konfiguraci a obsluhu tak, aby bylo možné na něm bez časově náročné konfigurace spustit i menší množství úloh.

1. Plánování dávkových úloh

1.1. Dávkové úlohy

Dávková úloha je úloha, která nevyžaduje žádnou interakci od uživatele. Jedná se o úlohu či skupinu úloh, které jsou vykonávány společně – v dávkovém módu. Dávková úloha má instrukce pro své zpracování uložené v souboru. Dávkové úlohy nepodporují uživatelskou interakci. [11,12,13]

Dávkové úlohy jsou často využívány pro běh programů náročných na výpočetní prostředky. Z tohoto důvodu jsou řazeny do fronty a plánovány tak, aby efektivně využily dostupné výpočetní prostředky. V případě plánování na určitý časový interval se nejčastěji jedná o noční hodiny, kdy systém není tak vytížený prací uživatelů. Často se také vykonávají na pozadí paralelně s interaktivními úlohami. Dále lze dávkové úlohy plánovat i periodicky, což najde využití např. v účetnictví – např. pro generování periodických účetních výpisů. [11,13]

Na desktopech se dávkové úlohy používají jako skripty – typickým příkladem této skupiny dávkových úloh je např. tisknutí více dokumentů na tiskárně. [12,13]

Vzhledem k významu dávkových úloh zahrnuje plánovače dávkových úloh řada operačních systémů, z nástrojů lze zmínit například daemon *Cron* pro Unixové systémy či *Task Scheduler* na operačních systémech od Microsoftu [13]. Dále existují nástroje pro plánování dávkových úloh v prostředí počítačových clusterů.

1.1. Plánování dávkových úloh v prostředí počítačových clusterů

Vzhledem k vysokému výkonu počítačových clusterů je možné tyto clustery použít na výpočet výpočetně náročných úloh. Avšak k plnému využití tohoto výkonu je nutné řešit problém efektivního využití výpočetních zdrojů. V kontextu dávkových úloh se jedná o jejich naplánování. Problémem, který se při plánování úloh řeší, je efektivní alokování prostředků úlohám a jejich následné vykonání. Tento problém se řeší pomocí dvou hlavních typů plánovacích algoritmů – pomocí plánovacích algoritmů dynamických a pomocí plánovacích algoritmů statických. [17, 18]

Hlavní rozdíl mezi těmito dvěma algoritmy je v počtu informací a času plánování, co požadují – statické plánovací algoritmy potřebují předem znát specifické informace, typicky se jedná o informace, jaké úlohy se budou vykonávat a informace dostupnosti procesoru. Podle těchto informací se poté plánuje spouštění úloh – a to buďto v době kompilace či v době spuštění plánovače. Dynamické algoritmy tyto informace předem neznají, a tak je nutné tyto úlohy plánovat až ve chvíli, kdy dorazí. [17, 18]

Oba typy algoritmů musí pro efektivní plánování musí řešit dále také specifické problémy v závislosti na typech úlohy – v tomto kontextu existují sériové a paralelní úlohy. Z tohoto důvodu se algoritmy plánování dále dělí na algoritmy plánování sériových úloh, algoritmy pro plánování paralelních úloh a na algoritmy pro plánování smíšených úloh. [17, 18]

2. Distribuovaný plánovač dávkových úloh

Prostředí počítačových uzlů je typicky víceuživatelské prostředí. Z tohoto důvodu je nutné pro efektivní využívání a sdílení takovýchto systémů dávkovými úlohami mít k dispozici aplikaci, která bude efektivně řídit spouštění těchto úloh a tím zajišťovat efektivní využití výpočetních zdrojů. Tímto programem je *Resource Manager*. [15]

Resource manager tyto úlohy pouští na základě definovaných plánovacích politik v závislosti na právě dostupných prostředcích a jejich předpokládaném využití. *Resource manager* se nestará pouze o čas spuštění úlohy, jeho úlohou je i určení, na jakém uzlu či clusteru bude vzhledem k ostatním úlohám vykonána. Dále se stará o rozdělení dostupných prostředků mezi všechny uživatele systému. [15]

Pro efektivní fungování systému plánovače musí tyto dávkové úlohy obsahovat dále obsahovat informace ohledně požadovaných výpočetních prostředků, různá nastavení pro řazení a prioritu ve frontě a poté informace nutné pro inicializaci běhového prostředí pro vykonání úlohy. Dále je nutné definovat různá časová omezení běhu úlohy. [15]

Moderní distribuované plánovače dávkových úloh jsou centralizované systémy, které v sobě implementují či přímo obsahují *Resource manager* [21]. Tyto systémy pro svou práci využívají fronty úloh, ze kterých postupně podle určitého algoritmu vybírají dávkové úlohy k vykonání. [17].

Algoritmy, podle kterých vybírají, jsou zaměřené na dosažení určitého optimalizačního cíle – tímto cílem může být například maximální využití prostředků, minimální čekací doba úloh, anebo maximalizace provedených úloh.

Plánovače v těchto algoritmech spoléhají především na uživatelsky nastavené atributy, které používají pro určení pořadí vykonávání – tento výpočet může probíhat pouze při zařazení anebo se může měnit i dynamicky [16]. Z těchto algoritmů je důležité zmínit *First Come First Served* (FCFS) a *Shortest Processing time First* (SPF). FCFS řadí podle času zařazení do fronty, SPF podle času očekávaného běhu [17].

Mimo klasického určování pořadí ve frontě podporují moderní plánovače i nejrůznější optimalizace, např. tzv. *backfilling*, což je technika, která umožňuje přiřazení systémových prostředků úloze s nižší prioritou a její vykonání mimo pořadí v případě, že toto vykonání nezpomalí vykonávání úloh s vyšší prioritou. [16, 17] Tyto techniky spoléhají zejména na očekávanou dobu vykonávání, kterou však plánovač většinou nedokáže odhadnout za běhu, a tak spoléhá na uživatelsky maximální dobu vykonávání – tyto zadané hodnoty jsou však značně nepřesné [17].

3. Nejznámější plánovače dávkových úloh

3.1. PBS Professional®

3.1.1. Základní informace

Projekt je k dispozici jako open-source řešení [1] nebo je k dispozici jako komerční řešení od firmy Altair [2]. Open-source řešení je známé pod názvem *OpenPBS*. Tento produkt je aktivně vyvíjen komunitou, neustále čerpá z vývoje komerčního řešení a je součástí *Linux Foundation*. Komerční řešení je známé jako *Altair® PBS Professional®*. Oba tyto projekty mají stejnou jednotnou dokumentaci dostupnou na komunitních stránkách Altair.

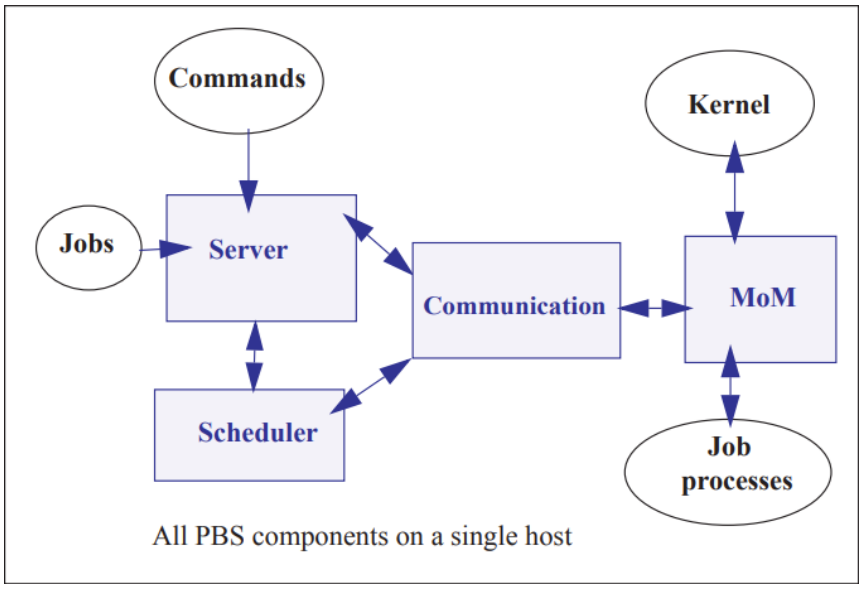
Dávkové úlohy jsou v aplikaci řízeny podle uživatelsky definovaných pravidel. Dávkové úlohy je možné spustit na vice clusterech, aplikace je škálovatelná a odolná proti selhání. [1, 2]

Aplikace je kompatibilní pouze s operačním systémem *Linux*, jediný *Job management daemon* (MoM) je kompatibilní s operačními systémy *Linux* a *Windows*. [3]

3.1.2. Struktura aplikace

Aplikace se skládá z příkazů, datové služby a daemonů, které navzájem tvoří systém. Tyto daemoni se dále dělí podle účelu na:

- *Server daemon* – daemon zodpovědný za centrální řízení,
- *Scheduler daemon* – daemon zodpovědný za plánování úloh,
- *Communication daemon* – daemon zodpovědný za správu komunikace systému,
- *Job management daemon (MoM)* – daemon zodpovědný za vykonávání úloh.



Obrázek 1 – Struktura PBS Professional [3]

3.2.2.1 Server

Server je daemon, který je zodpovědný za správu jednotlivých daemonů aplikace. Jedná se o centrální řídící prvek systému. Veškerá jeho komunikace s okolím probíhá přes síť pomocí protokolu IP. Tato komunikace probíhá na žádost jednotlivých klientů anebo na základě

monitorovaných změn. Dále je úlohou serveru ukládání čekajících úloh, posílání prací *MoM* k vykonání a monitoring jednotlivých prací s případnými restarty. Server veškeré úlohy řadí do front, které mají konfigurovatelné vlastnosti. [3]

3.2.2.2 Scheduler

Úkolem daemonu *Scheduler* je plánování úloh podle příslušné scheduling policy. Tato scheduling policy řídí, kde a kdy je spuštěna dávková úloha. Je možné nastavit i více schedulerů, každý z nich může mít svou vlastní scheduling policy. [3]

3.2.2.3 Job management daemon (MoM)

Daemon *MoM* je odpovědný za vykonávání dávkových úloh, jejichž kopie získává ze serveru. Daemon spustí dávkovou úlohu s co nejvěrnějším nastavením uživatelské relace. Poté co daemon vykoná úlohu, pošle výsledky na server. *MoM* si vede vlastní log, kam si zaznamenává informace spojené s vykonáváním úloh.

MoM také mohou jednu dávkovou úlohu sdílet, a tedy vykonávat tuto úlohu na více *MoM* současně, v tomto případě každá *MoM* vykonává část úlohy.

Na každém hostitelském počítači, co vykonává *PBS* dávkové soubory běží jeden *MoM* daemon. V závislosti na konfiguraci *MoM* vykonává aktivní polling, kdy se *MoM* dotazuje serveru na dostupné úlohy. [3]

3.2.2.1 Communication Daemon

Communication Daemon zodpovídá za komunikaci mezi ostatními daemony, kromě komunikace daemonu *Scheduler–Server* a *Server–Server*, které komunikují napřímo pomocí protokolu TCP. Komunikace daemonu s ostatními daemony probíhá pomocí protokolu TPP[1]. Daemon v sobě má zabudovaný mechanismus na znovu posílání informací, co příjemce nepotvrdil. Dále má v sobě daemon zabudovanou automatickou datovou kompresi, pomocí které automaticky komprimuje veškerá vyměňovaná data mezi ostatními daemony. [3]

3.1.3. Důležité entity v aplikaci

Z uživatelského pohledu pracuje aplikace s několika klíčovými entitami

1. *Vnode* – virtuální objekt, který reprezentuje skupinu výpočetních prostředků. Může se jednat o celého výpočetní klient, ale také pouze o jeho část. Každý klient musí být tvořen minimálně jedním vnodem. Všechny vnode jednoho klientova spravuje jedna *MoM*.
2. *Queue* – fronta pro veškeré příchozí joby. Pořadí v této frontě neurčuje pořadí vykonávání, jejím účelem je primárně evidenci jobů. Frontu lze založit s různou konfigurací a účelem – například pro typy jobů, pro různé vnody atd.
3. *Job* – jedná se o posloupnost příkazů, která se vykoná na výpočetních uzlech reprezentovaných vnode. Při nahrání jobu ho musíme nahrát kompletní i s jeho konfigurací – tzn. jaké výpočetní prostředky vyžaduje, do jaké fronty ho řadíme atd. [11]

3.1.4. Plánovací model

Plánování veškerých úloh probíhá pomocí plánovače úloh. *PBS* obsahuje celkem typy dva plánovačů – výchozí plánovač a multisched plánovač. Výchozí plánovač plánuje práce ve

[1] TCP-based Packet Protocol – protokol pro vytvoření trvalého TCP spojení využívaný v aplikaci PBS Professional® [3]

výchozí partition, multisched poté plánují joby ve svých nastavených partitions. Pro každý plánovač lze definovat plánovací logika.

Plánovací politika v *PBS Professional* je konfigurovatelná pomocí různých nástrojů *PBS*. Tyto nástroje jsou součástí scheduleru anebo i jiných entit v *PBS*. Součástí těchto nástrojů je určení priority jobu – tato priorita je separátně určovaná pro preempci a vykonávání jobů.

Výchozím chováním plánování v *PBS* je, že job umístí na vnode, kde jsou k dispozici prostředky, které job vyžaduje. [3]

1.1.4.1 Určení priority jobu

PBS Professional má v sobě zabudovaných několik nástrojů, které slouží k výpočtu priority jobu. Tyto nástroje umožňují mimo jiné řazení jobů do front, seskupování jobů do skupin, určení pořadí vykonávání těchto skupin, přesun joby mezi těmito skupinami. Dále je pomocí těchto nástrojů možno určit přesně, jaký job má navázat na job předchozí. Tyto nástroje lze různě kombinovat.

Nástroje lze rozdělit do několika níže uvedených skupin:

- Nástroje pro prioritizaci pomocí fronty,
- Nástroje pro řazení pomocí jobů,
- Nástroje pro prioritizaci pomocí času čekání,
- Nástroje pro výpočet politiky preempce.

Kromě těchto nástrojů prioritu jobu, a tedy i jeho následné pořadí vykovávání dále ovlivňuje časový slot – lze zvolit jeden ze tří: tzv. primetime, non-primetime a dedicated time. Dále prioritu ovlivňují zabudované systémy tříd jobu, které slouží například pro zohlednění rezervaci, preempci a přislušnosti jobu v expresní frontě.[3]

3.2. Slurm Workload Manager

3.2.1. Základní informace

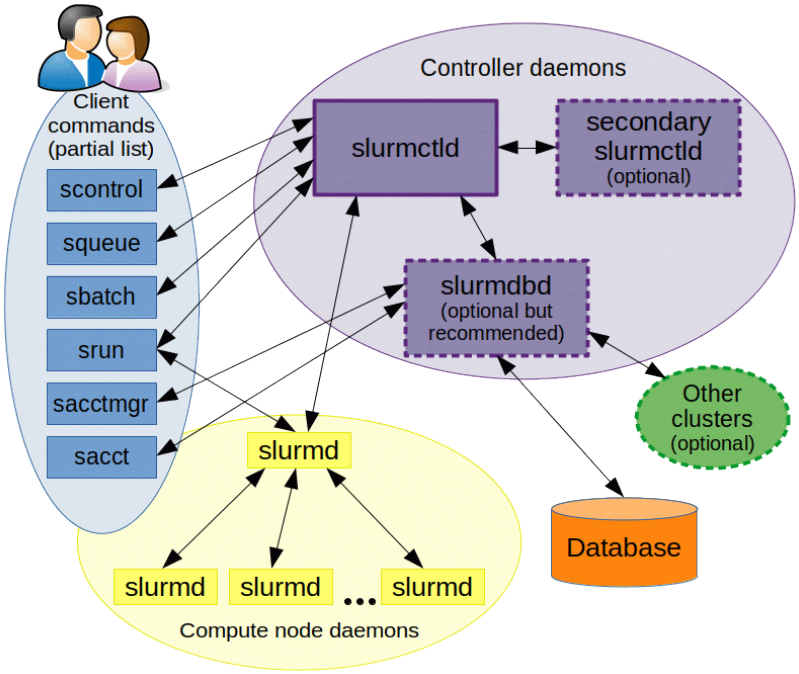
Software *Slurm* je open-source aplikací napsanou v jazyce C, jejíž vývoj vede společnost SchedMD za pomoci více než 200 firem a institucí z celého světa – jmenovitě se jedná například o společnosti a instituce Intel, HP, NVIDIA, National University of Defense Technology (NUDT, China) a další.

Aplikace *Slurm* se zaměřuje na rychlé, výpadku odolné a škálovatelné řešení v oblasti plánování dávkových úloh na výpočetních clusterech. *Slurm* je kompatibilní pouze s operačním systémem *Linux*. [4]

3.2.2. Struktura aplikace

Aplikace se skládá ze 2 povinných daemonů a 2 nepovinných daemonů. Povinné daemony jsou *slurmctld* daemon, který je centralizovaným správcem zdrojů a poté daemon *slurmd*, který se stará o přijímáni a vykonávání úloh na jednotlivých výpočetních uzlech. Nepovinným uzlem je daemon *slurmdbd*, který se stará o zápis informací ohledně využití clusterů spravovaných *Slurmem* do databáze. Další nepovinný daemon je daemon *slurmrestd* – ten umožňuje komunikaci se Slurmem pomocí REST API. Součástí Slurmu je také podpora pluginů, které jsou v praxi často využívány.

Slurm obsahuje také uživatelské nástroje jako jsou *srun*, *scancel*, *sinfo*, *squeue*, *sacct*, *sview*, *scontrol* a *sacctmgr*. [4]



Obrázek 2 – Struktura aplikace Slurm [4]

3.2.2.1 Slurmctld

Daemon *Slurmctld* je centrálním řídícím prvkem aplikace. Stará se o správu ostatních daemonů *Slurm*, monitoring dostupných prostředků, správu front s úlohami, správu úloh a s nimi spojených prostředků. Tento daemon je kritický pro fungování celé aplikace, proto je možné spustit paralelně i záložní *slurmctld* daemon, který v případě výpadku nahradí primární. [4, 5]

3.2.2.1 Slurmd

Slurmd se stará o správu výpočetních uzlů. Toto zahrnuje monitoring všech úloh, které jsou spuštěné na tomto uzlu, přijímání úloh, co tento uzel může zpracovat, samotné spouštění a běh úloh. Dále také zajišťuje robustní hierarchickou komunikaci s ostatními daemony. Daemon zabírá co nejméně místa a je optimalizovaný na efektivní využívání systémových prostředků. [4]

3.2.2.2 Slurmdbd

Volitelný daemon *Slurmdbd* je zodpovědný za správu přístupu k databázi *Slurmu*. Do této databáze ukládá daemon údaje o využití jednotlivých clusterů *Slurmu*. [4]

3.2.2.3 Slurmrestd

Volitelný daemon *Slurmrestd* poskytuje rozhraní, které umožňuje klientům komunikaci přes REST API. Toto rozhraní je synchronní a zpracovává požadavky pouze na jednom vlákně. V současné době podporuje dva módy – INET SERVICE MODE a LISTENING MODE. V INET SERVICE MODE se rozhraní chová jako Inet služba. V LISTENING MODE daemon poslouchá na příchozím portu a čeká na nová TCP spojení. I přes to, že *Slurm* REST API podporuje autentizaci a autorizaci pomocí JWT tokenů, není navrženo pro komunikaci přímo přes internet z toho důvodu, že pro komunikaci používá protokol HTTP. [4]

3.2.3. Důležité entity v aplikaci

Slurm z pohledu užívání pracuje s několika klíčovými entitami. Základní entitou ve Slurmu jsou *nodes*, *partitions*, *jobs* a *job steps*.

1. *Nodes* – výpočetní prvek, který má v závislosti na nastavení určité přidělitelné zdroje.
2. *Partition* – skupina tvořená Nodes, která se může i překrývat s ostatními partitions, v terminologii ostatních PBS se jedná o frontu (queue). Lze do nich přiřazovat Jobs
3. *Job* – jedná se o prvek, který na určitý čas alokuje výpočetní prostředky, aby je šlo použít pro spouštění příkazů a zpracování dat.
4. *Job step* – jednotlivé úlohy, které jsou prováděny v rámci job. Mohou být prováděny i paralelně.

3.2.4. Plánovací model

Základním chováním plánování na *Slurmu* je plánování pomocí fronty FIFO – tzn. joby jsou spuštěné ve stejném pořadí, v jakém byly nahrány. Za pomocí pluginů však lze nakonfigurovat řazení do fronty pomocí priority. Dále lze nakonfigurovat i další parametry plánování.

3.2.4.1 Výpočet priority a určení pořadí ve frontě

V prioritním plánování se joby plánují za pomocí spočítané priority, a i jiných faktorů, jmenovitě jde o tyto [9]:

- preemptivní pořadí (nahrazení jobu jiným jobem),
- rezervace v předstihu,
- priorita partition,
- priorita jobu,
- ID jobu.

Priorita jobu se spočítá následujícím vzorcem:

„$JobPriority = PriorityWeightAge \times AgeFactor + PriorityWeightFairshare \times FairShareFactor + PriorityWeightJobSize \times JobSizeFactor + PriorityWeightPartition \times PartitionFactor + PriorityWeightQOS \times QosFactor$“ [9]

Kde priorita úlohy (*JobPriority*) je číslo uložené ve formátu Integer 32 b, faktory (*Factor*) jsou čísla s plovoucí čárkou v rozsahu 0–1 a jednotlivé váhy priority (*PriorityWeight*) jsou čísla ve formátu Integer 32 b [9].

Vysvětlení pojmů použitých při výpočtu:

- *Age factor* – tato hodnota roste postupně s časem, který stráví job ve frontě. Do tohoto času se ovšem nepočítá čas, kdy job nemůže běžet kvůli nesplněným závislostem anebo kvůli vyjmutí z plánování.
- *Fair share factor* – tato hodnota se nastavuje s ohledem na férové obsloužení jobů z různých účtů.
- *Size factor* – tato hodnota se zvyšuje společně s velikostí alokovaných prostředků jobem.
- *Partition factor* – tato hodnota odráží prioritu partition, ve které se job nachází.
- *QOS factor* – tato hodnota závisí na nastavení priority dané QOS.

Jeden job může být přidělen i v několika dalších partitions současně, v tu chvíli se pro každou partition liší vypočítaná hodnota priority. V této situaci, ve chvíli, kdy je job vybrán a jsou mu alokovány prostředky, je jeho záznam v souvislosti s ostatními partition ignorován. Pokud je job znovu zařazen do fronty, tak se veškeré záznamy znovu zaktivní a job může být spuštěn na každé přidělené partition. [9]

3.2.4.2 Plánovací algoritmy

Slurm obsahuje několik typů plánovacích algoritmů, které svou výpočetní složitostí odpovídají tomu, jak často mají být spouštěny.

3.2.4.2.1 Quick Scheduling

Tento plánovací algoritmus je spuštěn událostmi, které zahrnují akce jako zadaní job do systému, dokončení job či změna konfigurace aplikace. V rámci tohoto algoritmu se projde pouze omezené množství jobů, určené konfigurací. Algoritmus zohledňuje i partitions – platí následující pravidlo: „Pokud je v nějaké části (partition) nějaká úloha ponechána jako čekající, žádné další úlohy v této části nejsou brány v úvahu pro plánování (tj. FIFO).“[2] [9].

3.2.4.2.2 Less Quick Scheduling

Jedná se o stejný algoritmus, jako quick scheduling, avšak provádí se každou minutu a postupně projde všemi joby ve frontě. Konec algoritmu je taktéž možný, pokud vyprší maximální nastavený čas procházení. Tento algoritmus tedy umožní spustit i joby umístěné v partitions, které mají menší prioritu.

3.2.4.2.3 Backfill Scheduling

Tento algoritmus má na starost spouštění jobů s menší prioritou ve chvíli, kdy jejich spuštění neposune očekávaný čas spuštění jobu s větší prioritou. V tomto algoritmu, aby fungoval správně, je nutné mít předem správný odhad konce běžících jobů, jelikož se z něj počítá výše zmíněný odhadovaný čas konce. Algoritmus prochází veškeré běžící joby, které vyhodnotí podle v pořadí podle priority a zjistí, kdy každý job začne a skončí s ohledem na preemtivnost, gang scheduling, paměťové nároky atd. Jakmile zjistí, že může spustit job, který právě prochází, bez toho, aniž by zdržel joby s větší prioritou, tak tak učiní, v opačném případě rezervuje jobu prostředky v očekávaném čase spuštění. [4]

3.2.4.2.4 Gang Scheduling

Slurm podporuje přiřazování stejných prostředků ve stejné partition více jobům naráz. Jedná se o přiřazování přes časové intervaly – to znamená, že vždy v jeden okamžik má pouze jeden z těchto jobů vyhrazený přístup k veškerým prostředkům. [4]

3.2.4.3 Preempce

Slurm umožňuje odebrání výpočetních prostředků jobu s menší prioritou za účelem spuštění jobu s větší prioritou. Pravděpodobnost preemce jobu závisí na konfiguraci systému (zejména na PriorityTier partition a QOS partition) a na vlastnostech jobu. V závislosti na nastavení lze také konfigurovat, co přesně preempce udělá s nahrazeným jobem – lze ho zrušit, spustit na jiných výpočetních prostředcích, pozastavit po dobu trvání jobu, který preempci spustil anebo lze také použít plánování Gang Scheduling. [4]

[2]Once any job in a partition is left pending, no other jobs in that partition are considered for scheduling (i.e. FIFO)

3.3. Vlastní řešení – porovnání s předešlými plánovači

Plánovač, vyvinutý v rámci této bakalářské práce je napsán ve vyšším programovacím jazyce Java. Jako grafické rozhraní bylo zvoleno rozhraní JavaFX. Aplikace sestává ze dvou modulů, jmenovitě se jedná o klient a server.

Vyvinutý plánovač byl inspirován mimo jiné i výše zmíněnými aplikacemi, avšak z důvodu jiného rozsahu a zaměření se mezi ním a těmito aplikacemi najde několik rozdílů. Srovnání těchto systémů s vyvinutým plánovačem je uvedeno v následující tabulce.

Tabulka 1 – Srovnání plánovačů úloh [autor]

Vlastnost	*PBS Professional®*	*Slurm*	*Vlastní řešení*
Komponenty	Server, scheduler, MoM a communication daemon	Controller daemoni, compute daemoni, database daemon, rozšiřující daemoni	Klient, server
Plánovací algoritmy	Plánování pomocí priority, možnost přizpůsobění pomocí konfigurace, podpora backfilling	Plánování pomocí priority, možnost přizpůsobění pomocí konfigurace a pluginů, podpora backfilling	Plánování za užití prioritní fronty s využitím více front s vlastní prioritou
Styl řízení	Centrální řízení, kde server zadává úlohu výpočetním uzlům	Centrální řízení, kde server zadává úlohu výpočetním uzlům	Centrální řízení, kde server zadává úlohu výpočetním uzlům
Dotazování výpočetních uzlů na úlohy	Aktivní dotazování klientů (polling) s možností interakce serveru a použití hooks	Klient prování aktivní dotazování (polling) na server	Klient prování aktivní dotazování (polling) na server
Podpora front či obdobných mechanismů	Routovací a vykonávající fronty, které umožňují uložení jednotlivých entit či skupin entit. Podle konfigurace pomáhají při určení priority úloh.	Partitions jako skupina výpočetních uzlů, mohou se navzájem překrývat, mají vliv na výpočet priority, jedna job může být součástí více partitions	Fronty s přiřazenou prioritou, fronta může být pouze jen v jedné frontě
Prostředky výpočetních uzlů	Vnodes, které reprezentují skupinu uzlů, mají určité výpočetní prostředky	Dělení na nodes, s pluginy mají nodes vlastní výpočetní prostředky jako např. CPU, Memory atd.	Virtuální jednotka reprezentující využité prostředky

	jako např. CPU, GPU atd.		
Podporované operační systémy	*Linux* a *Windows* (*Windows* pouze *MoM*)	*Linux*	*Linux* a *Windows*

4. Implementace vlastního systému

Cílem bakalářské práce je návrh a implementace multiplatformního distribuovaného síťového plánovače úloh, který umožňuje prioritní plánování a plánování pomocí fronty FIFO a poté následný sběr výsledků zpět na řídící uzel. Podle tohoto zadání bylo následně zpracováno zadání a požadavky na systém.

4.1. Zadání a požadavky systému

Smyslem a uplatněním tohoto plánovače je zrychlení a zefektivnění práce s dlouhotrvajícími výpočetně náročnými úlohami a možnost zadat na je jednom místě – vybraném centrálním serveru. Tyto úlohy se poté paralelně autonomně zpracují, bez nutnosti uživatelského zásahu, a vrátí požadované výsledky.

Zpracovávaný plánovač musí umět pracovat s dávkovými úlohami a musí umožňovat vykonat dávkové zpracování i s proměnnými parametry. Aplikace také musí podporovat minimálně operační systémy *Windows* a *Linux*.

Aplikace musí být typu klient – server, kde jeden server řídí *n* klientů, které se serveru dotazují pomocí pollingu. Server musí příchozí komunikaci přijímat na jediném portu, sám nesmí zahajovat odchozí komunikaci.

Zadávání úlohy serveru klientu musí být provedeno tak, že server klientu pošle kompletní aplikaci a veškeré informace o tom, s jakými parametry má být úloha spuštěna odkud mají být získány výsledky.

Také je kladen důraz na korektní vyřešení problému důvěryhodnosti a integrity přenášených dat.

Inspirací při návrhu systému byly aplikace *Slurm*, *PBS Pro*. Z těchto aplikací byly vytyčeny další cíle – systém musí podporovat práci s frontami a mít určitou míru odolnosti vůči chybám při zpracování úloh. V rámci této odolnosti je nutné evidovat i deadline klientů a úloh.

Z požadavků na implementaci systému a z vytyčených cílů tedy vyplývají následující funkční požadavky:

Funkční požadavky

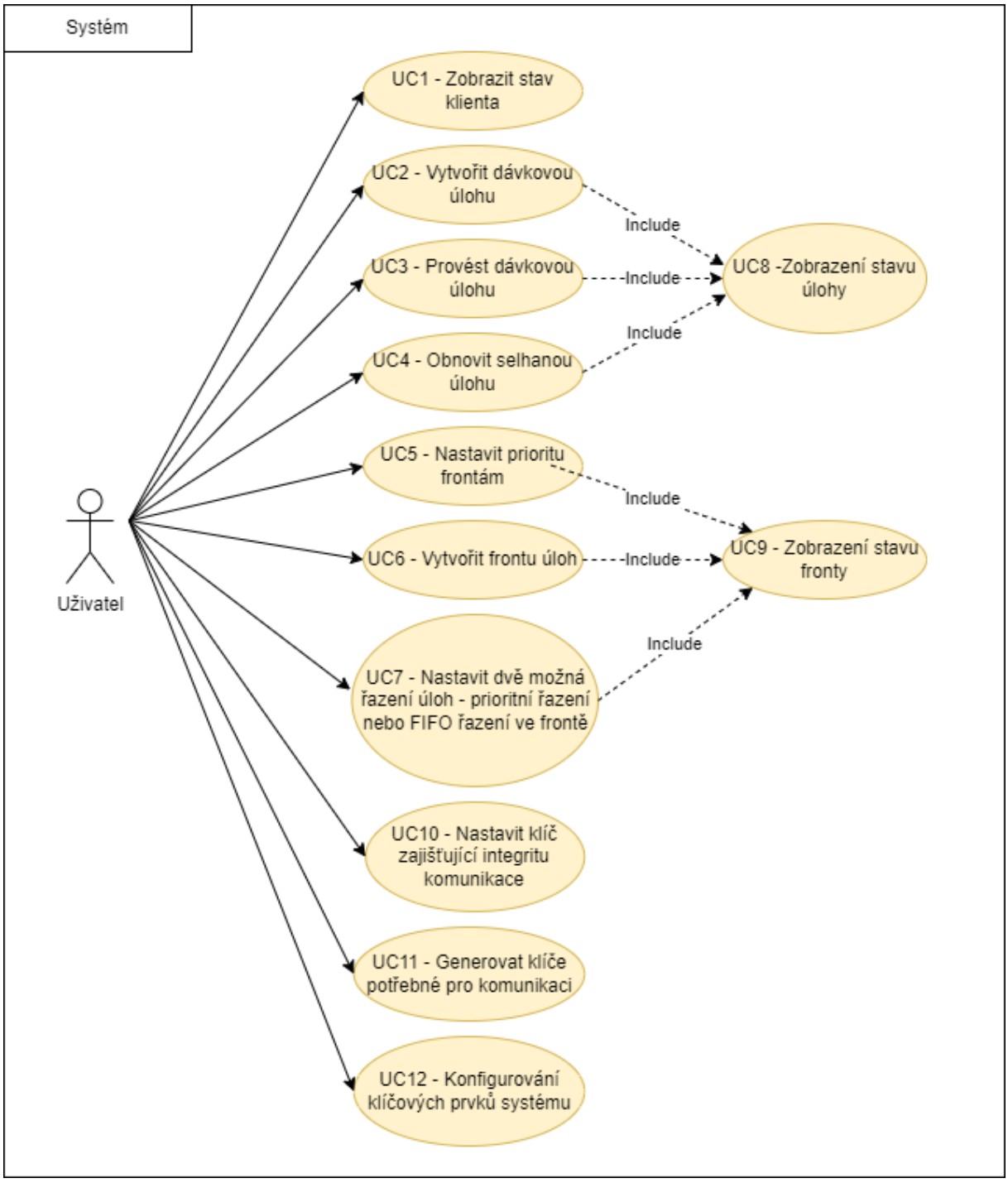
- R1 – Musí být umožněno zadávat úlohy ke zpracování.
- R2 – Úlohy musí být řazeny pomocí dvou algoritmů – prioritní plánování a FIFO.
- R3 – Aplikaci musí být možné konfigurovat.
- R4 – Aplikace musí umožňovat tvorbu front.
- R5 – Aplikace musí umožňovat generování klíčů.
- R6 – Aplikace musí umožňovat nastavení klíčů.
- R7 – Úlohy musí být zpracovány na klientu.
- R8 – Úlohy se odesílají na klienta celé včetně parametrů a cesty k výsledkům.

Nefunkční požadavky

- R9 **–** Aplikace musí podporovat minimálně operační systém Windows a Linux.
- R10 **–** Aplikace se musí skládat z aplikace klient a aplikace server.

- R11 – Aplikace server nesmí zahajovat odchozí aplikaci.
- R12 – Aplikace server naslouchá na jediném příchozím portu.
- R13 – Aplikace klient se serveru dotazuje pomocí pollingu.
- R14 – Zpracované úlohy musí být dávkové úlohy.
- R15 – Aplikace musí podporovat síťovou komunikaci mezi clustery.
- R16 – Aplikace musí kontrolovat deadline.
- R17 – Aplikace musí být odolná vůči chybám při běhu úlohy.

Z těchto požadavků byl odvozen use-case diagram, zobrazený na obrázku 3.

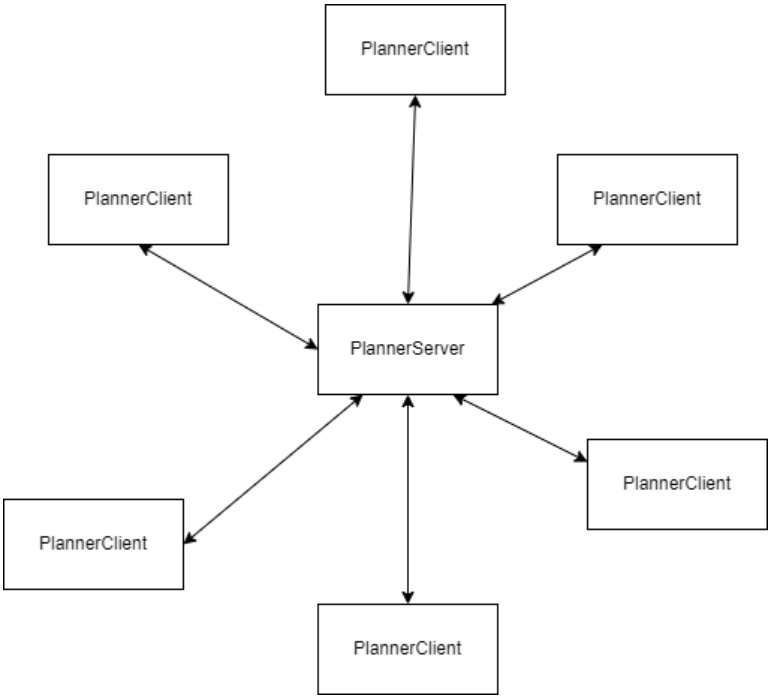


Obrázek 3 – Use-case diagram systému [autor]

4.2. Struktura aplikace

Systém plánovače se skládá ze dvou spolupracujících modulů – z modulu *PlannerClient* a z modulu *PlannerServer*. Oba moduly jsou aplikace, které jsou napsány ve vyšším programovacím jazyce Java.

Tyto moduly spolu komunikují za pomocí soketu. Tuto komunikaci vždy zahajuje modul *PlannerClient*. Pro kompletní spuštění všech funkcí systému je nutné spustit jednu aplikaci *PlannerServer* a 1 až *n* modulů *PlannerClient*.



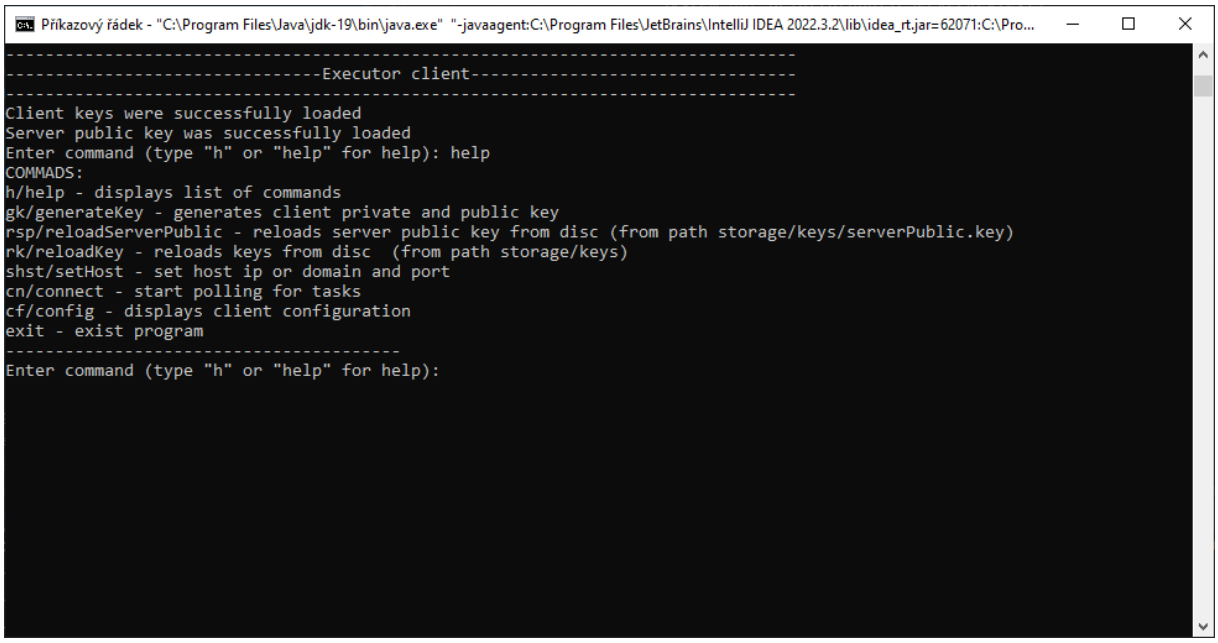
Obrázek 4 – Diagram systému se šesti spuštěnými klienty [autor]

4.3. Klient

Modul *PlannerClient* je výpočetním uzlem systému. Má na starosti korektní přijmutí dávkové úlohy ze serveru, jejího zpracování ve stanoveném čase a poté odeslání jejího výsledku zpět na server. Modul je implementován jako konzolová aplikace, která obsahuje ve svém rozhraní umožňuje zadat veškeré klíčové příkazy.

Seznam příkazů:

- h/help – nápověda,
- gk/generateKey – vygenerování dvojice veřejný a privátní klíč klienta,
- rsp/reloadServerPublic – přenačtení veřejného klíče uloženého v serverového modulu,
- rk/reloadKey – přenačtení dvojice veřejný a privátní klíč klienta,
- shst/setHost – nastaví adresu/hostname serverového modulu a příslušný komunikační port,
- cn/connect – naváže spojení se serverem pomocí pollingu,
- exit – ukončí program.



Obrázek 5 – Příkazy rozhraní modulu klient [autor]

Dále program obsahuje po zadání příkazu „connect“ příkazy „stop“ a „ps“ pro zastavení pollingu a pro vypsání úloh klienta.

4.3.1. Konfigurace

Aplikace PlannerClient umožňuje veškeré konfigurovatelné hodnoty nastavit pomocí konfiguračního souboru config.json. Tento soubor musí být uložen být uložen v kořenové složce aplikace. Soubor je čten pomocí knihovny Jackson, která se stará o jeho kompletní deserializaci a uložení do třídy Javy.

Následující příklad je plná funkční konfigurace, ve které jsou nastaveny veškeré hodnoty pro připojení klienta na server s adresou 127.0.0.1 pomocí portu 6660, konfigurovaný agent klienta je Windows, nastavené virtuální prostředky jsou 100 a fronty, ze kterých klient čerpá úlohy jsou "test Priority Queue","test FIFO"].

```
{
    "port": 6660,
    "host": "127.0.0.1",
    "agent": "WINDOWS",
    "subscribedQueues":["test Priority Queue","test FIFO"],
    "availableResources": 100
}
```

4.3.2. Autentizace

Autentizace a autorizace veškeré komunikace se serverovým modulem je zabezpečena pomocí privátních klíčů. Aplikace klient umožňuje generování těchto klíčů ve formátech PKCS#8 (privátní klíč) a X.509 (veřejný klíč). Algoritmus klíčů je RSA o velikosti 2048 b. Tyto klíče jsou generovány za pomocí standardního balíčku Javy java.security. Klíče jsou po vygenerování uloženy na disk (do adresáře storage/keys).

Z pevného disku jsou tyto klíče také načteny (ze storage/keys) společně s veřejným klíčem serveru, který musí být do této složky vložen pod názvem serverPublic.key. Klíč musí být pro správné přečtení ve formátu X.509.

4.3.3. Komunikace se serverem

Veškerá komunikace se serverem probíhá pomocí aktivního dotazování (polling). Toto dotazování probíhá z důvodu responzivity aplikace na jiných vláken, než na kterém běží hlavní úloha aplikace. Dotazování probíhá každých 100 ms a je realizováno pomocí skupiny vláken, která jsou udržována v poolu, který spravuje Executor service. Pro každou takto zahájenou komunikaci vždy vzniká nové vlákno, které má v případě obdržení úlohy na starost veškerý životní cyklus této úlohy.

Samotná komunikace je realizována za pomocí soketů, které vždy směřuje na nakonfigurovaný port. V případě, že se komunikaci se serverem nepodaří navázat, klient pokračuje v dotazování a vyčkává, dokud komunikaci nenaváže.

Po navázání spojení klient vygeneruje klíč pro asymetrickou komunikaci, který je ve formátu AES. Následně je validováno spojení se serverem formou challenge – response, a to za pomocí poslání náhodného řetězce, generovaného pomocí UUID, který je zašifrován klíče veřejným klíčem serveru. Shoda tohoto řetězce je poté validována s odpovědí od serveru, která musí přijít zašifrovaná veřejným klíčem klienta, ve formátu tohoto UUID. Následně klient obdrží výzvu ve formě textového řetězce (opět UUID), která je zašifrována jeho veřejným klíčem, tento řetězec musí klient správně přečíst a nezměněný odeslat zašifrovaný veřejným klíčem serveru. Po této autentizaci pošle klient serveru zašifrovaný AES klíč, pomocí kterého bude následující komunikace šifrována.

Po úvodní autentizaci dochází k registraci klienta na server anebo, pokud klient už id má, pokusí se tímto id představit. Dále klient dostane informaci, jestli dostane zadaný úkol. V případě, že klient nedostane žádný úkol, tak aktuální dotazovací vlákno je ukončeno. V opačném případě klient ověří, že na přijatý úkol má k dispozici dostatek výpočetních prostředků, v případě, že ano, pošle aktualizované dostupné prostředky a poté přistoupí k přijímání souborů úkolu. Přijatý úkol má formát zip a jedná se o celý dávkový soubor, který byl nahrán uživatelem. Po korektním přijmutí klient ukončí komunikaci a vykoná dávkovou úlohu.

Po vykonání úlohy dochází k navázání spojení za účelem předání výsledků serveru. Navázání a autentizace komunikace pro odeslání výsledků úlohy probíhá obdobně, jako navázání a autentizace v rámci pollingu. Po úspěšné autentizaci dochází k předání id a statusu úlohy serveru, který vyhodnotí, jestli je výsledek úlohy s tímto id očekáván od tohoto klienta. Pokud server odpoví negativně, tak je úloha smazána z klienta a je uzavřena komunikace. V případě kladné odpovědi dochází k přenosu výsledných dat na server. Tento přenos je opakován do té doby, dokud server nepošle finální odpověď.

Klient dále odesílá zprávu o chybě, pokud při vykonávání zadané úlohy dojde k chybě, a to pomocí navázání nového, obdobně autentizovaného spojení, kde po autentizaci dojde k odeslání ID chybně zpracované úlohy.

Zajímavostí je odesílání a přijímání šifrovaných zpráv v komunikaci. Zpracování textové zpráv je realizována pomocí zaslání zašifrovaného obsahu, který je dále kódován pomocí kódování base64 a na konec každé zprávy je přidán ukončovací symbol – ASCII kód 31.

Zpracování listu je založeno na podobném principu, jedná se o zařazení těchto textových zpráv do pole, poté zašifrování a zakódování tohoto pole pomocí kódování base64 a následné přidání koncového symbolu pole – ASCII kód 30.

Ukázka kódu sloužící pro odeslání šifrovaného textového obsahu

```
private void sendEncryptedString(Cipher aesEncrypting, DataOutputStream out, byte[]
messageToSend) throws IllegalBlockSizeException, BadPaddingException, IOException {
    String encrypted = encrypt(messageToSend, aesEncrypting);
    String withStop = encrypted + STOP_SYMBOL;

    out.write(withStop.getBytes(StandardCharsets.UTF_8));
    out.flush();
}

public String encrypt(byte[] decrypted, Cipher cipher) throws
IllegalBlockSizeException, BadPaddingException {
    byte[] cipherText = cipher.doFinal(decrypted);
    return Base64.getEncoder().encodeToString(cipherText);
}
```

Příjem těchto zpráv

```
private String readEncryptedString(Cipher aesDecrypting, InputStream in) throws
IOException, IllegalBlockSizeException, BadPaddingException {
    byte[] bytes = readBytesUntilStop(in, STOP_SYMBOL);
    String message = decrypt(bytes, aesDecrypting);
    return message;
}

private byte[] readBytesUntilStop(InputStream cipherInputStream,
char stopSymbol) throws IOException {
    List<Byte> bytes = new ArrayList<>();
    int byteAsInt = cipherInputStream.read();
    while (!(byteAsInt == -1 || (char) byteAsInt == stopSymbol)) {
        bytes.add((byte) byteAsInt);
        byteAsInt = cipherInputStream.read();
    }
    byte[] bytesArr = new byte[bytes.size()];
    for (int i = 0; i < bytesArr.length; i++) {
        bytesArr[i] = bytes.get(i);
    }

    return bytesArr;
}

public String decrypt(byte[] encrypted, Cipher cipher) throws
IllegalBlockSizeException, BadPaddingException {
    byte[] plainText = cipher.doFinal(Base64.getDecoder().decode(encrypted));
    return new String(plainText, StandardCharsets.UTF_8);
}
```

Ukázka kódu sloužící pro odeslání šifrovaného listu – metoda encrypt je shodná s metodou použitou při šifrování textových zpráv

```
public void sendEncryptedList(Cipher cipher, DataOutputStream out, List<String>
toEncrypt) throws IllegalBlockSizeException, BadPaddingException, IOException {
    String encrypted = encryptList(toEncrypt, cipher);
    out.write(encrypted.getBytes(StandardCharsets.UTF_8));
    out.flush();
}

public String encryptList(List<String> toEncrypt, Cipher cipher) throws
IllegalBlockSizeException, BadPaddingException {
    StringBuilder encodedArrWithDeli = new StringBuilder();
    for (String item : toEncrypt) {
        //encode each item
        String encodedItem = Base64.getEncoder()
         .encodeToString(item.getBytes(StandardCharsets.UTF_8));
        encodedArrWithDeli.append(encodedItem).append(STOP_SYMBOL);
    }
```

```
    //encrypt whole array
    String enryptedArr = encrypt(encodedArrWithDeli.toString()
    .getBytes(StandardCharsets.UTF_8), cipher);
    //add delimeter
    enryptedArr += (ARR_STOP_SYMBOL);
    return enryptedArr;
}
```

4.3.4. Zpracování přijatých dávkových úloh

Zpracování přijaté úlohy začíná jejím vybalením z archivu zip a uložením vybaleného souboru do složky se zadáním (storage/results). Složka, kam se archiv vybalí se před vybalením vyčistí. Poté se z příchozího souboru přečte veškerá konfigurace úlohy a sloučí se s informacemi, které byly o úloze přeneseny v rámci textové výměny. Následně dojde k validaci správného zadání proměnných hodnot konfigurace, a pokud je validace úspěšná, vyčistí se složka, kam budou na klientu dočasně uloženy výsledky.

Následně dojde k samotnému vykonání úlohy a poté k přesunutí výsledků do dočasné složky s výsledky. Součástí výsledků jsou i soubory standardního výstupu a chybového výstupu – tyto soubory jsou uloženy jako soubory .txt. V závislosti na výstupním kódu spuštěného procesu dojde i k nastavení příslušného stavu úlohy. Tyto výsledky jsou poté zabaleny do archivu zip a připraveny k odeslání.

4.4. Server

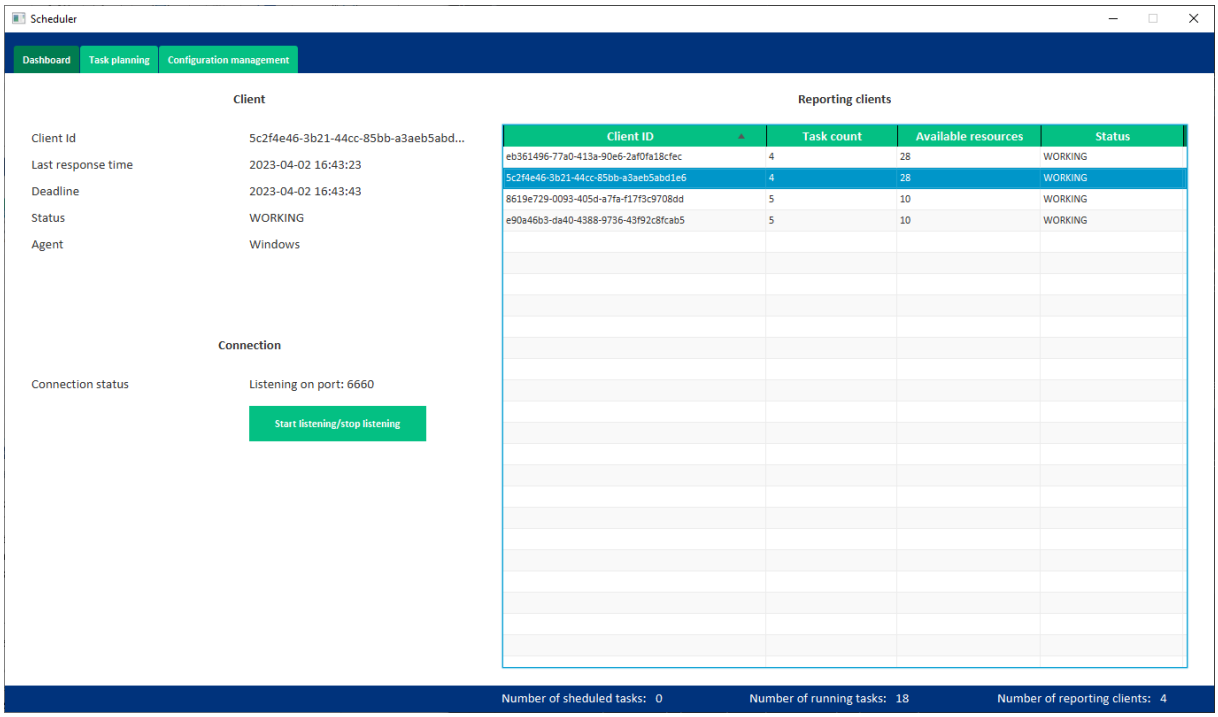
Modul *server* je řídícím prvkem systému. Jeho úlohou v systému je plánování úloh na jednotlivé klienty, správa stavu úloh, přijímání úloh od uživatele, jejich odeslání na výpočetní uzel a následné přijmutí výsledků. Dále tento modul také spravuje připojené klienty a kontroluje veškeré deadline.

Server je realizován jako aplikace s grafickým rozhraním v JavaFX.

4.4.1. Komponenty grafického rozhraní

4.3.1.1 Dashboard

Na dashboardu je zobrazen přehled všech evidovaných klientů společně s klíčovými informacemi o jejich stavu. Dále se zde zobrazuje informace o stavu naslouchání klienta na portu. Také zde lze tento stav přepnout za pomoci tlačítka.



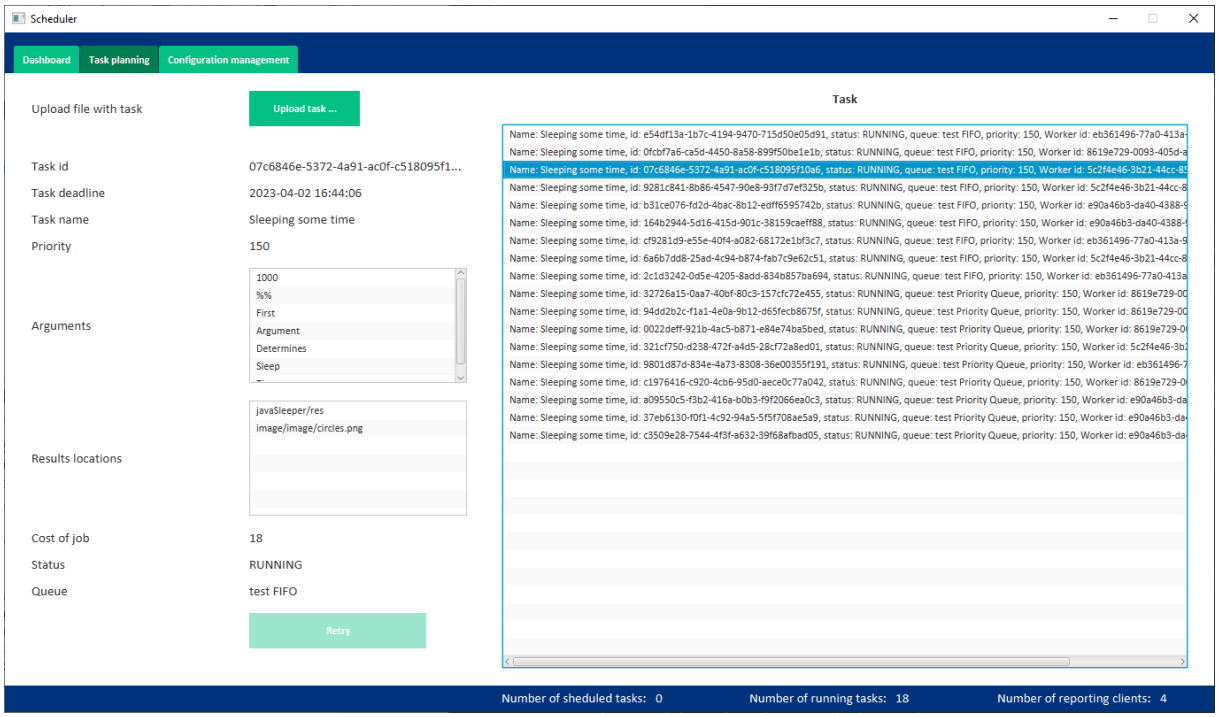
Obrázek 6 – GUI aplikace – panel dashboard [autor]

K zobrazení tabulky klientů byla využita komponenta JavaFX TableView, do které jsou data dodávána pomocí listu, který je periodicky obnovován pomocí úlohy observer, která je periodicky prováděna na vlákně v pozadí. Pomocí této úlohy jsou obnovovány i veškeré informace o vybraném klientovi, tyto informace jsou také navíc obnovovány i při každé změně vybrané položky. Obnova informace o naslouchání serveru je vázána pouze na stisk tlačítka.

4.3.1.2 Task planning

Tato sekce aplikace slouží pro zobrazení stavu všech zadaných úloh. Dále zde lze za úlohy nahrávat a také restartovat úlohy, které během vykonávání selhaly.

Nahrání souboru přes tlačítko upload je možné vždy (samotné nahráni však nemusí skončit úspěšně), restartování úlohy je možné pouze, pokud je vybraná úloha ve stavu FAILED a fronta, do které tato úloha patří, existuje a má dostatečnou kapacitu.



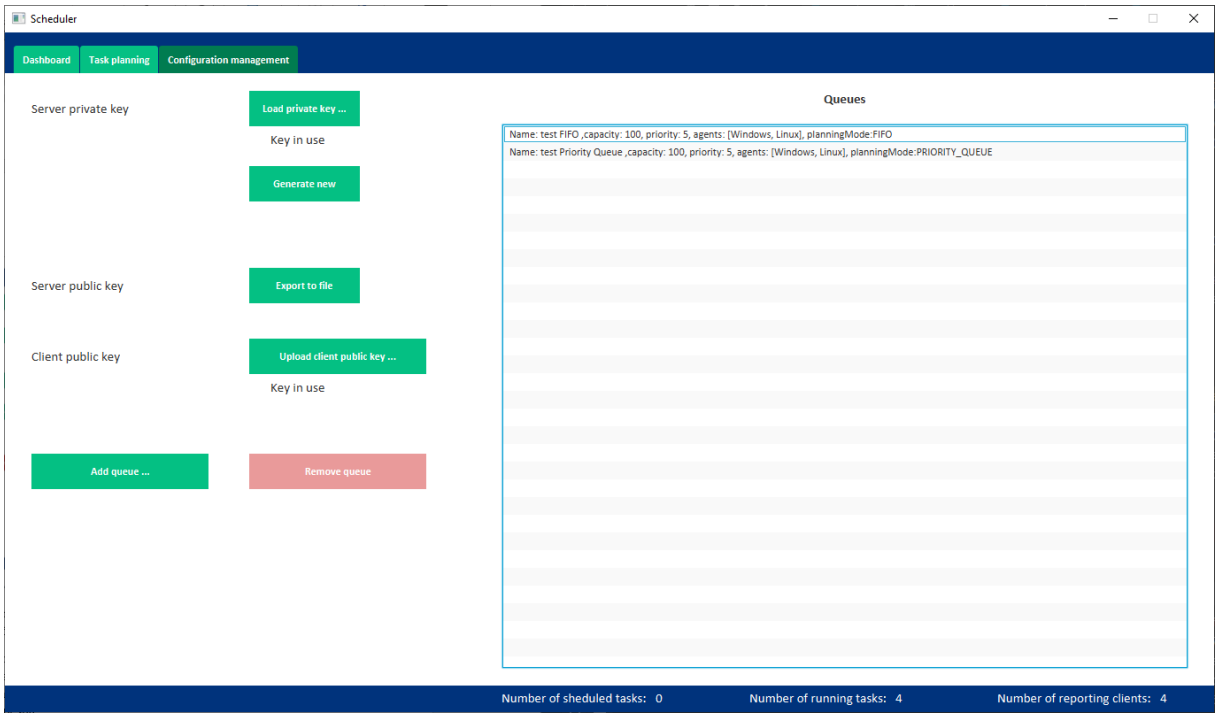
Obrázek 7 – GUI aplikace – panel Task Planning [autor]

List všech úloh je komponenta JavaFX listView, jejíž zdrojový list je obnovován ze seznamu již dokonaných úloh a také ze všech seznamů úloh všech front. Podseznamy ke každé úloze jsou také listView, zdroj každého z nich čerpá z příslušného neměnného listu.

Samotné obnovování listu je realizováno při každé změně stavu jakékoliv úlohy. Obnovování informací a podseznamů jednotlivé úlohy je navázáno na výběr v hlavním listu, obnovuje se společně s hlavním listem a také při změně výběru.

4.3.1.3 Configuration management

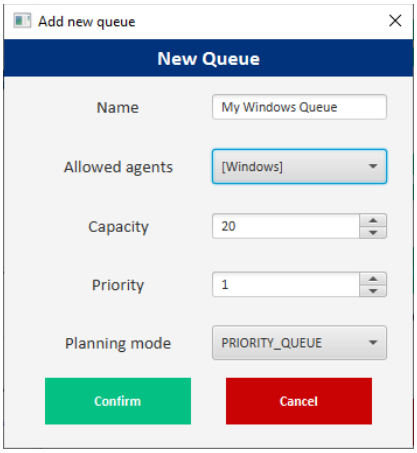
Cílem této sekce je poskytnutí informací o autorizačním klíči klientů a autorizačním klíči serveru a také správa jednotlivých front úloh. V této sekci se nachází tlačítka pro načtení privátního klíče ze serveru, vygenerování nové dvojice veřejný/privátní klíč, export veřejného klíče do souboru a také nahrání veřejného klíče klienta. Dále lze pomocí tlačítek spustit dialog pro přidání fronty a také frontu odebrat. Všechna tlačítka, až na tlačítko pro odebrání fronty lze spustit za jakýchkoliv podmínek, tlačítko pro odebrání fronty je aktivní pouze pokud je vybrána fronta a tato fronta neobsahuje žádné běžící úlohy.



Obrázek 8 – GUI aplikace – panel Configuration Management [autor]

Dostupné fronty jsou čerpány z mapy front a zobrazovány v seznamu ListView. Tento seznam je aktualizován při přidání či odebrání fronty.

Samotné přidávání fronty probíhá ve novém dialogovém okně. V tomto okně lze nastavit jméno fronty, povolené typy uživatelských agentů, kapacitu fronty, její prioritu a způsob plánování úloh. Tlačítko pro potvrzení obsahuje validaci na správné vyplnění údajů a zaktivní se pouze tehdy, pokud je validace úspěšná.



Obrázek 9 – GUI aplikace - přidávání nové fronty [autor]

4.3.1.4 Dolní stavová lišta

Aplikace dále obsahuje v dolní části stavovou lištu, která je nezávislá na aktuálně zvolené sekci, na které se nacházejí informace o počtu aktuálně naplánovaných úloh, o počtu aktuálně prováděných úloh a dále o počtu aktuálně evidovaných klientů.

4.4.2. Implementace backendu

4.3.2.1 Konfigurace

Server umožňuje konfiguraci z konfiguračního souboru, který se musí nacházet v kořenovém adresáři aplikace a jehož jméno musí být config.json. Z této konfigurace lze načíst list front, komunikační port, deadliny pro komunikaci s klientem a defaultní přidaný čas, po který je tolerováno vypršení deadline úlohy. Tento soubor je čtený pomocí open-source knihovny Jackson, která se stará o veškerou deserializaci dat a jejich uložení do příslušné třídy. Z této třídy jsou poté hodnoty přečteny při inicializaci aplikace. V případě, že není aplikace nalezena anebo je konfigurační soubor ve špatném formátu, jsou použity výchozí hodnoty.

```
{
    "queues":[
        {
            "name":"test Priority Queue",
            "agents":["WINDOWS","LINUX"],
            "planningMode":"PRIORITY_QUEUE",
            "capacity": 100,
            "priority": 5
        },
        {
            "name":"test FIFO",
            "agents":["WINDOWS","LINUX"],
            "planningMode":"FIFO",
            "capacity": 100,
            "priority": 5
        }
    ],
    "port":6660,
    "clientTimeoutDeadline": 20000,
    "clientNoResponseTime":2000,
    "taskTimeoutDelay":2000
}
```

4.3.2.2 Autentizace

Autentizace serveru je zajištěna obdobným způsobem, jako autentizace na klientu. Probíhá pomocí dvojice veřejný a privátní klíč. Server umožňuje generování této dvojice ve dvou formátech, a to jsou formát PKCS#8 pro privátní klíč a formát X.509 pro veřejný klíč. Jejich algoritmus je RSA s velikostí 2048 b. Klíče jsou obdobně jako u klientu uloženy na disk do storage/keys. Pro správnou funkci aplikace je také nutné do této složky nahrát také veřejný klíč klienta – ten musí být stejný pro všechny klienty.

Veřejný klíč klienta se do složky storage/keys nemusí nahrávat manuálně. Aplikace umožňuje klíč nahrát pomocí příslušného tlačítka grafického rozhraní – ve výše zmíněném formátu PKCS#8. Dále aplikace umožňuje export veřejného klíče serveru pomocí prvků grafického rozhraní. Klíč je vyexportován do zvolené složky ve formátu PKCS#8 pod zvoleným jménem.

4.3.2.3 Nahrávání úloh

Úlohy se nahrávají skrze grafické rozhraní. Každá nahraná úloha musí obsahovat konfigurační soubor ve formátu json se jménem taskConfig.json, který obsahuje konfiguraci úlohy. Dále je nutné, aby úloha obsahovala složku payload, ve které se nachází soubory s úlohou.

Implementace samotného nahrání spočívá v přečtení složky vybrané z grafického rozhraní a poté následnou validací, zda jsou nahrány všechny nezbytné soubory, zda jsou správně vyplněny všechny požadované hodnoty a zda je k dispozici příslušná fronta, která musí mít

dostatečnou kapacitu. O stavu validací je uživatel informován. Pokud jsou všechny validace v pořádku, úkol se registruje do fronty na seznam nenaplánovaných úloh, poté dojde na vlákně běžícím na pozadí k jeho zabalení do archivu zip a po zabalení se přesune ze seznamu nenaplánovaných úloh do plánovací fronty.

4.3.2.4 Plánování úloh

Plánování v aplikaci probíhá za pomocí front. Výběr probíhá v závislosti na odebíraných frontách klientem a následně algoritmu dané fronty. Na začátku plánování se nejprve vyberou všechny odebírané fronty a setřídí se sestupně podle své priority. Následně se se prochází seznam těchto front z vrchu, a z tohoto seznamu se vždy vyberou fronty se stejnou prioritou. Poté se z každé takto vybraných front vybere úkol podle plánovací strategie fronty. Pokud byl vybrán alespoň jeden úkol, tak se jednotlivé úkoly se porovnají mezi sebou a vybere úkol s nejvyšší prioritou, na který má klient k dispozici prostředky. Pokud úkol nebyl nalezen anebo mají všechny tyto úkoly cenu vyšší, než jsou dostupné prostředky klienta, výběr front se opakuje. Pokud ani po projití všech úloh nebyl nalezen odpovídající úkol, tak byl výběr neúspěšný.

Následně je nalezený úkol přidělen klientu a dále je klientu odesláno id a cena úkolu. Pokud klient potvrdí, že má dostupné prostředky, je úkol přidělen úspěšně, pokud ne, tak se úkol opět vrátí do příslušné fronty a výběr byl neúspěšný.

Algoritmy pro plánování úloh jsou v aplikaci implementovány dva. Prvním algoritmem je algoritmus prioritní fronty. Úlohy ve frontě implementující tento algoritmus se řadí a vybírají striktně podle své priority s tím, že přednost mají vždy úlohy s nejvyšší prioritou. Druhým algoritmem je pak algoritmus FIFO (First In First Out). Pokud má fronta nastavený tento algoritmus, tak se úlohy řadí a provádí podle pořadí vložení.

4.3.2.5 Fronty

Aplikace umožňuje řazení úloh do front. Každá z těchto front má svou vlastní prioritu a svůj vlastní plánovací algoritmus. Dále má fronta určitou kapacitu. Fronta lze vytvořit vždy, smazat lze pouze, pokud neobsahuje žádné úlohy.

Uložení úloh ve frontě je realizováno pomocí 3 kolekcí – z toho 2 jsou plánovací a vždy se zvolí pouze jedna z nich podle zvoleného plánovacího algoritmu. Třetí kolekcí je list nenaplánovaných úloh, který je na plánovacím algoritmu nezávislý.

4.3.2.6 Kontrola termínů

Aplikace umožňuje kontrolu termínů deadline pro běžící úlohy a také dva deadline pro evidované klienty. Kontrola je realizována pomocí vlákna, které periodicky každých 200 ms porovnává u každé úlohy a klienta čas deadline. Pokud je čas deadline a nastavený čas tolerované překročení doby deadline u úlohy, tato úloha selže. U klienta dojde při překročení jeho prvního deadline k označení za neodpovídajícího klienta, při překročení druhého deadline a tolerované doby dojde k jeho odstranění ze seznamu klientů. V případě odstranění dále dojde k odebrání jeho úloh a následné označení úloh za selhané.

4.3.2.7 Komunikace s výpočetními uzly

Komunikace s výpočetními uzly probíhá na jednom, předem známém konfigurovatelném portu a vždy ji zahajuje výpočetní uzel – klient. Server vždy pouze naslouchá na příchozím portu. Veškerá komunikace probíhá za pomocí soketů.

Na začátku komunikace server obdrží od klienta výzvu – náhodný text ve formátu uuid, který je zašifrovaný veřejným klíčem serveru. Server tento klíč přečte a odešle nezměněný text, který zašifruje veřejným klíčem klienta. Poté server pošle obdobnou výzvu klientu – také zašifrovaný náhodný text ve formátu uuid. V tomto případě je poslaný text zašifrován veřejným klíčem klienta. Server následně čeká na odpověď zašifrovanou svým veřejným klíčem, po rozšifrování se text musí shodovat s textem výzvy, v opačném případě server spojení ukončí.

Po této úvodní autentizaci server obdrží zašifrovaný AES klíč, který je následně použit v komunikaci pro její zašifrování. Server následně čeká na identifikaci klienta. V závislosti na zprávě, kterou klient pošle, tak server zaregistruje nového klienta, identifikuje stávajícího klienta, identifikuje žádost o předání výsledku anebo zprávu o selhání úlohy.

V případě registrace nového klienta dojde ke zjištění jeho aktuálních prostředků, jeho odebíraných front a jeho agenta. Id a aktuální prostředky se zjistí z původní žádosti, fronty se zjistí z listu, který klient dále pošle. Po úspěšném přečtení těchto údajů přidělí server klientu unikátní identifikátor uuid a odešle mu ho. Tato registrace probíhá synchronně, tím je zajištěno, že klient je registrován pouze jednou.

Ve chvíli, kdy se klient identifikuje, se zjišťuje id, pod kterým se identifikuje. Pokud se toto id shoduje s některým záznamem na serveru, klient je identifikován a je aktualizován záznam o jeho dostupných prostředcích, který poslal společně se svým identifikátorem. Avšak může nastat i situace, že na serveru příslušný identifikátor není nalezen (např. pokud byla aplikace serveru ukončena a poté byla spuštěna její nová instance). Tato situace je vyřešena tím, že je klient registrován obdobným způsobem, jako v případě nové registrace.

V případě registrace nebo identifikace se dále přistupuje k zadání úkolu klientovi, tento algoritmus je popsaný výše v kapitole plánování úloh.

Pokud server obdrží požadavek s žádostí o předání výsledku, ověří si nejprve, že klient, který tuto žádost posílá je evidován na serveru – pokud není, vytvoří se na serveru nový záznam a klientu se pošle odpověď obsahující nové id. Následně dojde ke kontrole, že klient má přidělený alespoň jeden úkol. Poté se z příchozí zprávy zjistí id úkolu a podle něj se úkol dohledá v seznamu úkolů klienta. Následuje samotné přijmutí výsledku – výsledky se ukládají do adresáře storage/results. Výsledek se přijímá ve formě archivu zip. Přijmutí výsledku se v případě chyby opakuje – maximálně však třikrát. Podle úspěchu či neúspěchu se klientu poté pošle správa se statusem, na serveru se u úkolu tento status nastaví. V případě úspěchu následně dojde k vyčištění složky se zipem vstupních souborů dávkové úlohy a úloha je označena za dokončenou.

Další možností je, že server obdrží od klienta s žádostí o předání chybového stavu úlohy. Server si v tu chvíli ověří, obdobně jako v předchozím případě, že klient je evidován a popřípadě ho registruje. Poté server zjistí, jestli úkol s přijatým id eviduje v seznamu úkolů tohoto klienta. Pokud takový úkol eviduje, tak ho nastaví jako selhaný.

4.5. Klíčové entity rozeznávané v aplikaci

V systému je rozeznáváno množství klíčových entit, které slouží k zajištění správného fungování systému. Nejdůležitějšími entitami v aplikaci jsou entity client, task a queue. Tyto entity budou dále popsány z pohledu serverové části aplikace.

4.5.1. Client

Tato entita slouží pro reprezentaci připojeného výpočetního uzlu k serveru – tedy jedna instance této entity odpovídá jednomu modulu *PlannerClient* připojenému k serveru. U této entity jsou evidovány informace nutné pro korektní správu výpočetního uzlu. Jedná se o konfigurační informace – id klienta, o jeho evidovaný operační systém, poslední zjištěné výpočetní prostředky klienta, jeho status z pohledu serveru, čas posledního navázaného spojení a odebírané fronty klientem. Dále jsou evidovány informace o úlohách, které byly klientu zadány a aktuálně je provádí.

4.5.2. Task

Task je entita zodpovědná za reprezentaci zadané dávkové úlohy. Entita eviduje atributy nutné pro správu životního cyklu úlohy – od zadání uživatelem po přijmutí výsledků. Tato entita získává většinu svých konfiguračních atributů z konfiguračního souboru payload.json. Mezi tyto atributy patří název úlohy, cena jejího provádění, spuštěný příkaz, parametry spuštěného příkazu, cesta, ze které se získávají výsledky, priorita úlohy, wildcard parametr, hodnota wildcard parametru, čas timeoutu a fronta, do které úloha patří. Dále se u úlohy eviduje id, klient, který úlohu provádí, začátek spuštění, cesta ke zdrojové složce, cesta k jejímu zipu a její status.

4.5.3. Queue

Queue je entita zodpovědná za reprezentaci fronty úloh, kam se řadí veškeré úlohy. Tato entita obsahuje konfigurační atributy – to jsou jméno, povolené operační systémy klientů, kapacita, informace o plánovacím módu a vlastní priorita. Dále se zde nachází fronty úloh – prioritní fronta a fronta FIFO. Úlohy se nahrávají a spouští vždy z jedné z těchto front v závislosti na plánovacím módu. Entita také obsahuje frontu nenaplánovaných úloh sloužící pro ukládání aktuálně nahrávaných úloh a běžících úloh.

4.6. Možnosti dalšího rozvoje aplikace

Možností vývoje aplikace nad rámec zadání této práce je několik. Veškeré další zmíněná potenciální rozšíření jsou inspirována aplikacemi *Slurm* a *PBS Professional®.*

Jako jedna z nejdůležitějších možností se jeví rozvoj algoritmu plánování v souladu s plánovacími algoritmy aplikací *Slurm* a *PBS Professional®.* Z tohoto pro rozvoj je nejdůležitější podpora plánování ve víceuživatelském prostředí a zní vyplývající faktory ovlivňující prioritizaci služeb jednotlivých uživatelů (např. implementace QOS) a jejich implementace do již implementovaných algoritmů FIFO a prioritního plánování. V případě tohoto rozšíření by bylo nutné provést zásadní změny – například upravit aktuální serverovou část aplikace na webovou aplikaci – tedy nahradit stávající grafické rozhraní za rozhraní webové aplikace, přidat databázi pro ukládání informací o jednotlivých uživatelích, do této databáze přesunout i informace o konfiguraci a dále přidat i rozšířit stávající systém autorizace o autorizaci jednotlivých uživatelů. V případě rozšíření o víceuživatelské prostředí se jeví jako

vhodné aplikaci rozšířit také o logování spuštěných úloh pro účely analýzy využití uzlů jednotlivými uživateli.

Dalším přínosným rozšířením plánování se jeví podpora algoritmů a technik zvyšující výkon a využití koncových uzlů, jako jsou například gang scheduling a backfill scheduling. Pro správnou funkci těchto algoritmů je dále nutno do aplikace při budoucím vývoji nahradit stávající systém virtuálních prostředků uzlů prostředky, které blíže odpovídají skutečné architektuře koncových uzlů – například pomocí implementace systému nodes podobného systému v aplikaci *Slurm*.

ZÁVĚR

Cílem této bakalářské práce je návrh a implementace aplikací síťového plánovače a exekutoru dávkových úloh v prostředí počítačových clusterů. Dále byla cílem práce rešerše nejznámějších nástrojů, které slouží k obdobnému účelu.

Součástí analýzy a návrhu aplikace je vymezení funkčních a nefunkčních požadavků aplikace. Dále je v rámci analýzy vypracován use-case diagram.

V dále jsou v teoretické části popsány veškeré grafické a logické části jednotlivých aplikací – jedná se o tzv. High-level design aplikace. Grafické moduly obsahují popis funkcionality z pohledu uživatele, u vybraných částí, zejména části zobrazující kolekce, se jedná i o popis použitých komponent grafického rozhraní. Logické části aplikace obsahují informace o algoritmickém průběhu jednotlivých funkcionalit v rámci systému. Poté jsou popsány klíčové entity aplikace. Dále je v aplikaci shrnut další možný rozvoj aplikace.

Implementovaná aplikace je inspirována nástroji zmíněnými v rešerši, jedná se v podstatě o jejich zjednodušenou verzi. Systém splňuje veškeré požadavky, které jsou součástí zadání bakalářské práce. Systém je tedy tvořen dvěma aplikacemi, které jsou síťové aplikace a jejich model komunikace je typu klient-server.

Server v souladu se zadáním poslouchá na jediném příchozím portu, na kterém se ho aplikace typu klient aktivně dotazují. Server může řídit 1 až *n* klientů. Od uživatele tato část systému přijímá dávkové úlohy, jejichž vykonávání plánuje podle algoritmů FIFO a prioritního plánování, v závislosti na plánu poté předává úlohu ke zpracování a po provedení úloh klientem získává výsledky zpět. Pro splnění požadavku multiplatformnosti je aplikace napsána v jazyce Java a v grafickém rozhraní JavaFX.

Klient je konzolová aplikace, která v souladu se zadáním provádí aktivní dotazování serveru a pomocí tohoto dotazování komunikuje se serverem. Od serveru přijímá celou dávkovou úlohu s parametry, tu vykoná a předá serveru výsledky. Klient je pro splnění požadavku přenositelnosti napsán v programovacím jazyce Java.

V souladu se zadáním práce je ve vypracovaném systému také řešen problém integrity a důvěrnosti přenášených dat. Tento problém je řešen pomocí šifrování komunikace. Komunikace je šifrovaná z počátku asymetrickou kryptografií – dvojicí veřejný/privátní klíč, poté pomocí symetrické kryptografie.

V rešerši obdobných systémů jsou představeny systémy *Slurm* a *PBS Professional*®. Jsou představeny základní informace o systémech, informace o jejich architektuře, klíčové entity v aplikacích, jejich plánovací modely a způsoby plánování. Dále je provedeno srovnání těchto systémů a jejich porovnání s v této práci implementovanou aplikací.

POUŽITÁ LITERATURA

[1] OpenPBS Open Source Project. OpenPBS Open Source Project [online]. Copyright © Copyright 2023 PBS Works is a division of Altair Engineering Inc., currently listed on Nasdaq as ALTR. All Rights Reserved. [cit. 08.03.2023]. Dostupné z: https://www.openpbs.org/

[2] Workload Manager and Job Scheduler for HPC | Altair PBS Professional. Altair | Discover Continuously. Advance Infinitely – Only Forward. [online]. [cit. 08.03.2023]. Dostupné z: https://altair.com/pbs-professional

[3] PBS Professional 2022.1.0 Big Book [online]. Copyright © Altair Engineering Inc. [cit. 08.03.2023]. Dostupné z: https://help.altair.com/2022.1.0/PBS%20Professional/PBS2022.1.pdf

[4] Slurm Workload Manager – Documentation. Slurm Workload Manager – Documentation [online]. Dostupné z: https://slurm.schedmd.com/

[5] Manuál k ubuntu [online]. Copyright © 2019 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd. [cit. 11.03.2023]. Dostupné z: https://manpages.ubuntu.com/

[6] Introduction to Slurm, Part 2 - YouTube. YouTube [online]. Copyright © 2023 Google LLC [cit. 11.03.2023]. Dostupné z: https://www.youtube.com/watch?v=LJrY0AthLB8&ab_channel=SchedMDSlurm

[7] Reading Academic Computing Cluster – Slurm commands and resource allocation policy – Academic Computing Team. [online]. Dostupné z: https://research.reading.ac.uk/act/knowledgebase/racc-slurm-commands-and-resource-allocation-policy/

[8] Slurm Batch Jobs | crc.pitt.edu | University of Pittsburgh. crc.pitt.edu | University of Pittsburgh [online]. Dostupné z: https://crc.pitt.edu/user-support/jobs-and-slurm-workload-manager/slurm-batch-jobs

[9] JETTE, Morris ,Tuning Slurm Scheduling for Optimal Responsiveness and Utilization [online]. [cit. 08.04.2023]. Dostupné z: https://slurm.schedmd.com/SUG14/sched_tutorial.p–f

[10] MESITI, Michele. BRIEF OVERVIEW OF THE SLURM JOB SCHEDULING ALGORITHM [online]. 2020 [cit. 2023-03-12]. Dostupné z: https://mmesiti.github.io/slurmprio200428/

[11] Submitting, Inspecting and Cancelling PBS Jobs – HPC Wiki. | High Performance Computing Center [online]. Dostupné z: https://hpccenter.sns.it/page/wiki/pages/Submitting_Inspecting_and_Cancelling_PBS_Jobs.html

[12] About batch jobs. Indiana University Knowledge Base [online]. Copyright © [cit. 15.03.2023]. Dostupné z: https://kb.iu.edu/d/afrx

[13] What is a Batch Job? EasyTechJunkie [online]. Dostupné z: https://www.easytechjunkie.com/what-is-a-batch-job.htm

[14] What Is a Batch Job? – BMC Software | Blogs. BMC Software – Run and Reinvent [online]. Copyright ©Copyright 2005 [cit. 15.03.2023]. Dostupné z: https://www.bmc.com/blogs/batch-jobs/

[15] Batch Jobs under LSF 10. [online]. Copyright © 2023 [cit. 15.03.2023]. Dostupné z: https://www.hpc.dtu.dk/?page_id=1416

[16] ZHANG, Di, Dong DAI, Youbiao HE a Bing XIE. RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning. ArXiv.org [online]. Ithaca: Cornell University Library, arXiv.org, 2020 [cit. 2023-03-16].

[17] ZRIGUI, Salah, Raphael Y. DE CAMARGO, Arnaud LEGRAND a Denis TRYSTRAM. Improving the performance of batch schedulers using online job runtime classification. Journal of parallel and distributed computing [online]. Elsevier, 2022, 164, 83-95 [cit. 2023-03-18]. ISSN 0743-7315. Dostupné z: doi:10.1016/j.jpdc.2022.01.003

[18] BARBOSA, Jorge a Moreira BELMIRO. Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters [online]. 2011 [cit. 2023-03-26]. Dostupné z: doi:https://doi.org/10.1016/j.parco.2010.12.004

[19] What is high-performance computing (HPC)? Purchase Intent Data for Enterprise Tech Sales and Marketing – TechTarget [online]. Dostupné z: https://www.techtarget.com/searchdatacenter/definition/high-performance-computing-HPC

[20] History and overview of high performance computing [online]. [cit. 08.04.2023]. Dostupné z: https://www.math-cs.gordon.edu/courses/cps343/presentations/History_and_Overview_of_HPC.pdf

[21] Understand job scheduler and resource manager [online]. [cit. 2023-04-10]. Dostupné z: https://www.icer.msu.edu/sites/default/files/files/understand_job_scheduler_v2.pdf