**EKONOMICKÁ UNIVERZITA V BRATISLAVE**
**FAKULTA HOSPODÁRSKEJ INFORMATIKY**

Evidenčné číslo: 103004/I/2025/36124048425840132

# OPTIMALIZÁCIA VÝKONU DISTRIBUOVANÉHO DATABÁZOVÉHO SYSTÉMU MYSQL CLUSTER

**Diplomová práca**

**2025** **Bc. David Birošík**

**EKONOMICKÁ UNIVERZITA V BRATISLAVE**
**FAKULTA HOSPODÁRSKEJ INFORMATIKY**

# OPTIMALIZÁCIA VÝKONU DISTRIBUOVANÉHO DATABÁZOVÉHO SYSTÉMU MYSQL CLUSTER

**Diplomová práca**

**Študijný program:** Informačný manažment

**Študijný odbor:** Ekonómia a manažment

**Školiace pracovisko:** Katedra aplikovanej informatiky

**Vedúci záverečnej práce:** doc. Ing. Mgr. Peter Schmidt, PhD.

**Bratislava 2025** **Bc. David Birošík**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky



# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc. David Birošík
**Študijný program:** informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
**Študijný odbor:** ekonómia a manažment
**Typ záverečnej práce:** Inžinierska záverečná práca
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** Optimalizácia výkonu distribuovaného databázového systému MySQL Cluster

**Anotácia:** Diplomová práca sa zaoberá komplexnou analýzou výkonnostných charakteristík MySQL Clusteru, distribuovaného databázového systému navrhnutého pre zabezpečenie vysokého výkonu a dostupnosti. Práca identifikuje hlavné výkonnostné obmedzenia a bottlenecky systému pri spracovaní real-time dátových požiadaviek. Na základe analýzy sú navrhnuté a implementované optimalizačné techniky vrátane pokročilého indexovania, efektívneho cachovania a optimalizácie konfigurácie siete. Tieto stratégie sú následne testované v kontrolovanom prostredí na súbore realistických workloadov, aby sa overil ich vplyv na zlepšenie priepustnosti a latencie systému. Práca tiež porovnáva výkon MySQL Clusteru pred a po aplikácii optimalizačných opatrení a poskytuje odporúčania pre ich implementáciu v produkčných prostrediach.
Praktická práca

**Vedúci:** doc. Ing. Mgr. Peter Schmidt, PhD.
**Katedra:** KAI FHI - Katedra aplikovanej informatiky
**Vedúci katedry:** doc. Ing. Mgr. Peter Schmidt, PhD.

**Dátum zadania:** 28.02.2024

**Dátum schválenia:** 11.03.2024 prof. Ing. Ivan Brezina, CSc.
osoba zodpovedná za realizáciu študijného programu

**Čestné prehlásenie**

Čestne prehlasujem, že som diplomovú prácu s názvom: Optimalizácia výkonu distribuovaného databázového systému MySQL Cluster, vypracoval samostatne pod vedením svojho školiteľa: doc. Ing. Mgr. Peter Schmidt, PhD.

V Bratislave, dňa: 15. apríla 2025 _____

Bc. David Birošík

**POĎAKOVANIE**

Týmto by som sa chcel poďakovať môjmu školiteľovi a vedúcemu diplomovej práce doc. Ing. Mgr. Petrovi Schmidtovi, PhD. za odborné rady a pomoc ktorou prispel k vypracovaniu tejto diplomovej práce.

**ABSTRAKT**

BIROŠÍK, David: *Optimalizácia výkonu distribuovaného databázového systému MySQL Cluster*. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Vedúci záverečnej práce: doc. Ing. Mgr. Peter Schmidt, PhD. – Bratislava: FHI EU, 2025, 73 s.

Cieľom záverečnej práce bolo vytvoriť distribuovaný databázový systém, ktorý bol optimalizovaný pre zabezpečenie vysokého výkonu a dostupnosti. Databázové servery v rámci distribuovaného systému využívali technológiu MySQL a samotný distribuovaný databázový systém bol vytvorený na technológii MySQL Galera Cluster. Jednotlivé časti práce boli zamerané na predstavenie teoretických konceptov databázových systémov ako napríklad možností ich konfigurácie, nasadenia ako distribuovaného systému alebo testovania výkonu. Avšak práca ponúka aj širší pohľad na koncepty sieťového spojenia a zabezpečovania vysokej dostupnosti aplikácii, ktoré je možné implementovať v distribuovanom databázovom systéme. Okrem teoretických konceptov sa práca pozerala aj na implementáciu týchto konceptov do distribuovaného databázového systému. Výsledkom riešenia danej problematiky bol funkčný návrh distribuovaného databázového systému, ktorý bol optimalizovaný na sieťovej úrovni ale aj na úrovni databázového systému.

**Kľúčové slová:** distribuovaný systém, databázový systém, optimalizácia výkonu

**ABSTRACT**

BIROŠÍK, David: *Optimizing the performance of the MySQL Cluster distributed database system.* – University of Economics in Bratislava. Faculty of Economic Informatics. Department of Applied Informatics. – Thesis Supervisor: Ing. Mgr. Peter Schmidt, PhD. – Bratislava: FHI EU, 2025, 73 s.

This masters thesis focused on creating database cluster, that was optimised in regard to high performance and availability. Database servers which created the cluster were using MySQL database technology. Connecting them into single cluster was done using cluster technology - MySQL Galera Cluster. Individual parts of this thesis were focused on theoretical introduction to concepts of database systems. For example, configuration of databases, deployment of cluster or performance testing. However, this thesis also offered a broader view on concepts of networking or high-availability solutions that were possible to implement in database cluster. As a result, this thesis offered functional proposal of database cluster, that was optimised on networking layer but also on database layer.

**Key words:** distributed system, database system, performance optimalisation

# OBSAH

# Zoznam diagramov



# Zoznam grafov

# Zoznam obrázkov



# Zoznam tabuliek

# Úvod

Počiatky databáz a potrebu uchovávať údaje môžeme nájsť už v ďalekej minulosti pred vznikom PC a moderných technológii. Stačí sa pozrieť na historickú civilizáciu Egypťanov, ktorí využívali hlinené dosky na sledovanie toku peňazí alebo surovín v ich civilizácii (Harford, 2017). Postupom času sa dáta začali uchovávať v obrovských kartotékach, čo však spôsobovalo komplikácie s vyhľadávaním dát, hľadaním súvislostí a vytváraním analýz na základe dát (Vierling, 2023).

V roku 1960, Charles W. Bachman vytvoril a integroval prvý databázový systém. Krátko na to sa objavil aj databázový systém od spoločnosti IBM. Avšak za začiatok databáz ako ich poznáme môžeme považovať rok 1970 v ktorom Edgar F. Codd zverejnil publikáciu *A Relational Model for Data for Large Shared Banks*. Táto publikácia opisovala RDBMS systém, ktorý umožňoval efektívne ukladanie dát a možnosť odpovedať na množstvo otázok, pokým databáza obsahovala potrebné dáta (Vierling, 2023). Neskôr v 70 rokoch 20 storočia dvaja vedci v IBM – Donald Chamberlin a Raymond Boyce vytvorili koncept SQL databáz. SQL databázy sa stali komerčne dostupné v roku 1979 a sú využívané do dnes (Mucci, 2024). Aj napriek tomu, že ide o staršiu technológiu, SQL databázy v roku 2019 predstavovali viac ako 60% databázových systémov vo svete (ScaleGrid Inc., 2019).

Dnes predstavuje technológia *MySQL* približne 38 % databázových systémov (ScaleGrid Inc., 2019). História *MySQL* siaha do roku 1996, kedy bola vydaná prvá verzia *MySQL* 1.0 iba pre limitovanú skupinu užívateľov. Prvá verejne dostupná verzia *MySQL* 3.11.1 bola zverejnená v októbri 1996 pre *OS* Solaris. O mesiac neskôr prišlo ku zverejneniu verzie aj pre OS založené na *Linuxe* (Pachev, 2007).

Vzhľadom na rastúce množstvo dát a potrebnú výpočtovú technológiu na prevádzku a spracovanie informácii sa spoločnosti rozhodli rozdeliť výpočtovo náročné operácie do menších častí na viacero serveroch. Týmto smerom postupovali aj pri databázach, čo zapríčinilo vznik distribuovaných databázových systémov (Kusnetzky, 2017).

# 1 Súčasný stav riešenej problematiky doma

Táto kapitola slúži na priblíženie rôznej funkčnosti databázových systémov a sieťových technológii. Kapitola sa primárne zaoberá pojmami a technológiami, ktoré boli využité v neskorších kapitolách práce, pri optimalizácii distribuovaného databázového systému.

## 1.1 Databázové systémy

Na trhu existuje veľké množstvo databázových systémov, ktoré majú svoje jedinečné využitie. Môžeme sa stretnúť s rôznymi databázami založenými na SQL, ako napríklad *MariaDB*, *MySQL*, *PostgreSQL* alebo aj inými, ktoré nepracujú s SQL ako napríklad *Redis*, *KeyDB*, *MongoDB*, atď. V tejto časti práce sme sa pozreli na tri rôzne databázové systémy, ktoré sme využívali naprieč touto prácou.

### *1.1.1 MySQL*

Databázový systém MySQL vznikol v roku 1996, kedy bola uvedená jeho prvá verzia, avšak verejne sa stal dostupným až koncom daného roka (Pachev, 2007). Podľa *StackOverflow Developer Survey 2024* je druhým najpoužívanejším databázovým systémom, hneď po *PostgreSQL*. Až 39.4% percenta respondentov sa v tomto prieskume vyjadrilo, že značná časť ich práce s databázovými systémami bola práve na systéme *MySQL* (Stack Exchange Inc., n. d.). Do roku 2008 sa na vývoji podieľala spoločnosť *MySQL AB*, ktorá však v roku 2008 bola odkúpená spoločnosťou *Sun Microsystems*. Aktuálne sa na jeho vývoji podieľa spoločnosť *Oracle*, ktorá v roku 2009 odkúpila spoločnosť *Sun Microsystems* (Descamps, 2024).

Systém *MySQL* je využívaný malými podnikmi, prípadne blogmi prevádzkovanými na redakčných systémoch ako napríklad *WordPress* ale využívajú ho aj nadnárodné spoločnosti ako napríklad *Meta* (*Facebook*), *X* (ex. *Twitter*), *Booking.com* alebo *Uber* (Descamps, 2024).

### *1.1.2 MariaDB*

Vzhľadom na obavy o budúcnosti *MySQL* po oznámení odkúpenia spoločnosťou Oracle, pôvodný zakladateľ Michael Widenius spolu s väčšinou pôvodných vývojárov *MySQL* využil pôvodný zdrojový kód *MySQL* a vytvoril alternatívnu verziu s názvom *MariaDB* (MariaDB Foundation, n. d.). *MariaDB* je do veľkej miery plnohodnotným ekvivalentom *MySQL* avšak postupom vývoja prišlo ku rozdielnym rozhodnutiam čo

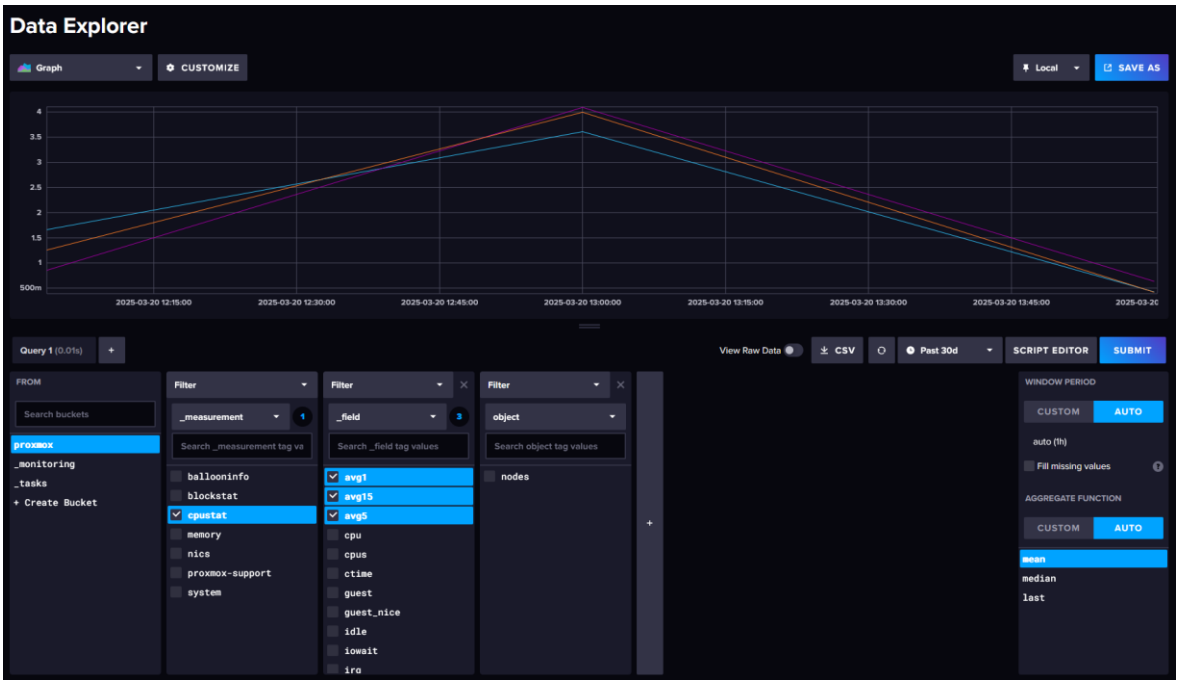
zapríčinilo, že niektoré funkcie nie sú kompatibilné medzi *MySQL* a *MariaDB* (MariaDB, 2025).

Rovnako ako pri *MySQL* tak aj pri *MariaDB* platí, že je využívaná naprieč širokým spektrom používateľov – od blogerov až po spoločnosti ako *Google* alebo *Wikipedia*.

### *1.1.3 InfluxDB*

Databázový systém *InfluxDB* je odlišný oproti predchádzajúcim systémom. Ide o databázový systém, ktorý je úzko špecifikovaný na ukladanie veľkého počtu časových údajov (IONOS editorial team, 2023). Pod týmto pojmom si môžeme predstaviť rôzne údaje zo senzorov, IoT zariadení a podobne. Nehovoríme teda o relačnej databáze ako v prípade *MySQL* alebo *MariaDB*. V tejto práci budeme databázu *InfluxDB* využívať na ukladanie dát z rôznych monitorovacích výstupov našich serverov počas vykonávania testov.

*Obrázok 1: Dáta vizualizované v systéme InfluxDB*




Zdroj: vlastné spracovanie

## 1.2 Spôsoby replikácie distribuovaných databáz

Pri distribuovaných databázových systémoch je možné využívať viacero druhov replikácie, ktoré zabezpečujú prenos databázových údajov z jedného uzla databázového systému na ostatné. Tieto spôsoby spolu môžu byť prepojené a vytvárať jeden systém na replikáciu dát medzi uzlami.

### 1.2.1 Binárne záznamy

Binárne záznamy sú komponentom, ktorý uchováva všetky zmeny v databáze. Ide o záznamy, ktoré umožňujú sledovať všetky zmeny, ktoré boli vykonané v databáze (Samuel, 2025). Keďže uchovávajú všetky zmeny, je možné ich využiť aj na replikáciu databázy na iný uzol v rámci distribuovaného systému. Pri *MySQL Galera Clusteri* môžu byť tieto záznamy využité pri prvotnom pripájaní uzla do distribuovaného systému alebo pri obnove uzla po jeho výpadku.

Uchovávané môžu byť v troch rôznych formátoch – *STATEMENT*, *ROW* a *MIXED*. V prípade *STATEMENT* spôsobu uchovávania sú do záznamov pridávané iba príkazy, ktoré vykonávajú zmeny v databázach a sú deterministické ako napríklad *UPDATE*, *DELETE* ,atď. Ide o najstarší formát binárnych záznamov. *ROW* formát nemá obmedzenia v podobe nutnosti deterministických výsledkov SQL dopytov. Zapisuje zmeny v dátach, nie príkazy ktoré sú spustené, čo má aj svoje nevýhody. Ak SQL dopyt vykonáva zmeny vo veľkom počte riadkov, tak binárny záznam bude zaberať veľa diskového priestoru. *MIXED* formát predstavuje kombináciu predchádzajúcich formátov. Ak je príkaz vyhodnotený ako deterministický, je zapísaný v *STATEMENT* formáte, naopak ak nie je deterministický, zapíše sa v *ROW* formáte (Razzoli, 2014).

### 1.2.2 GTID

GTID (*global transaction identifier*) je unikátny identifikátor spojený s každou transakciou na počiatočnom serveri. Tento identifikátor sa skladá z dvoch častí id zdroja a id transakcie, ktoré je poradovým číslom sekvencie vykonaných transakcií (Pattanayak, 2022). Túto techniku využíva MySQL replikácia, ktorá kopíruje dáta z hlavnej databázy na sekundárnu.

### 1.2.3 Wsrep a Galera Replication plugin

Na binárne záznamy a GTID záznamy nadväzuje wsrep. Ide o API vrstvu, ktorá je súčasťou *MySQL* a *MariaDB Galera Clusteru*. Táto vrstva používa replikačný model, ktorý považuje databázu za stavovú. Stav databázy zodpovedá jej obsahu. Pri zmene obsahu databázy sa zmení aj jej stav a wsrep API reprezentuje tieto zmeny ako transakcie (Codership Ltd, n. d.).

*Galera Replication plugin* implementuje túto API a na jej základe umožňuje vytvorenie distribuovaných databázových systémov, ktoré majú viacero primárnych uzlov (Codership Ltd, n. d.).

## 1.3 Konfigurácia databázových systémov

Databázové systémy majú veľké množstvo konfiguračných nastavení, za pomoci ktorých je možné optimalizovať a prispôsobovať správanie sa systému. Umožňujú nám prispôsobiť databázu napríklad na prevádzku v rámci zdieľaného prostredia, kedy sa na jednom serveri nachádza veľké množstvo databáz s rôznymi potrebami alebo optimalizovať za špecifickými cieľmi. V tejto kapitole sa pozrieme na premenné, ktoré sme neskôr v práci upravovali a optimalizovali. Ich vysvetlenia sme čerpali z oficiálnej dokumentácie na stránkach *MySQL* (Oracle Corporation, 2025).

### *1.3.1 innodb_redo_log_capacity*

Premenná definuje akú veľkú kapacitu disku majú zaberať tzv. *redo logs*. Tieto záznamy sú využívané v prípade, že príde ku pádu databázového servera a slúžia na obnovu nedokončených transakcií. Predvolene na disku zaberajú približne 9MB.

### *1.3.2 innodb_buffer_pool_size*

Určuje veľkosť pamäte RAM, ktorá obsahuje dočasne uložené dáta z *InnoDB* tabuliek a ich indexov. Jej predvolená hodnota je 125MB pamäte RAM, avšak pri systémoch s väčšou systémovou pamäťou sa odporúča jej navýšenie na čo najväčšiu hodnotu. Samotná dokumentácia *MySQL* odporúča na serveri vyhradenom iba na prevádzku databáz vyhradiť až 80% pamäte RAM za týmto účelom.

### *1.3.3 innodb_io_capacity*

Na základe tejto premennej je možné špecifikovať limit operácii písania a čítania za sekundu, tzv. IOPS, dostupných pre *InnoDB* úlohy na pozadí. Jej predvolená hodnota je 10000 IOPS. V prípade zmeny je pri tejto premennej potrebné brať do úvahy rýchlosť diskového poľa servera.

### *1.3.4 innodb_read_io_threads a Innodb_write_io_threads*

Tieto premenné špecifikujú počet systémových vláken pre operácie čítania a zápisu do InnoDB databázovej štruktúry. Jej predvolená hodnota sa vypočíta ako:

$$hodnota = \frac{počet\ logických\ procesorov\ servera}{2}$$

$$hodnota \geq 4$$

Na starších verziách *MySQL* (pred verziou *MySQL 8.4*) mala predvolenú hodnotu 4.

### *1.3.5 wsrep_slave_threads*

Ide o premennú špecifickú pre *MySQL / MariaDB Galera Cluster*. Definuje počet vláken pre proces replikácie a umožňuje paralelný zápis dát do databázy. Predvolená hodnota je 1 vlákno, avšak pri dostatočnom výkone servera je možné hodnotu navýšiť až na 512 vláken (Codership Oy, 2025).

### *1.3.6 innodb_log_buffer_size a innodb_log_file_size*

Veľkosť *innodb_log_buffer_size* nám udáva priestor v pamäti RAM, ktorý *InnoDB* systém využíva na uchovávanie záznamových súborov. Tieto súbory sú ukladané na disk a premenná *innodb_log_file_size* nám určuje miesto na disku, ktoré môžu zaberať.

V prípade *innodb_log_buffer_size* je predvolená hodnota 64MB pamäte RAM a pri *innodb_log_file_size* 48MB diskového priestoru. Väčšia hodnota *innodb_log_buffer_size* umožňuje udržať v pamäti RAM väčšie množstvo transakcií, čo redukuje počet IO operácii vykonávaných nad diskovými poliami.

### *1.3.7 innodb_flush_log_at_trx_commit*

Premenná určuje ako často sú zmeny v databázovom systéme zapisované na disk. Môže nadobúdať tri hodnoty – „0“, „1“ a „2“. Pri hodnote „0“ sú záznamy ukladané na disk každú sekundu, pri hodnote „1“ (predvolená hodnota pre premennú) sú záznamy ukladané na disk po každej transakcii. Hodnota „2“ predstavuje strednú cestu – záznamy sú zapisované na disk po každej transakcii, avšak ukladané sú iba raz za sekundu. Zmena tohto nastavenia z predvolených hodnôt predstavuje možnosť straty dát v prípade výpadku databázového systému. Pri distribuovanom systéme, vzhľadom na prítomnosť viacerých uzlov, je toto riziko minimalizované.

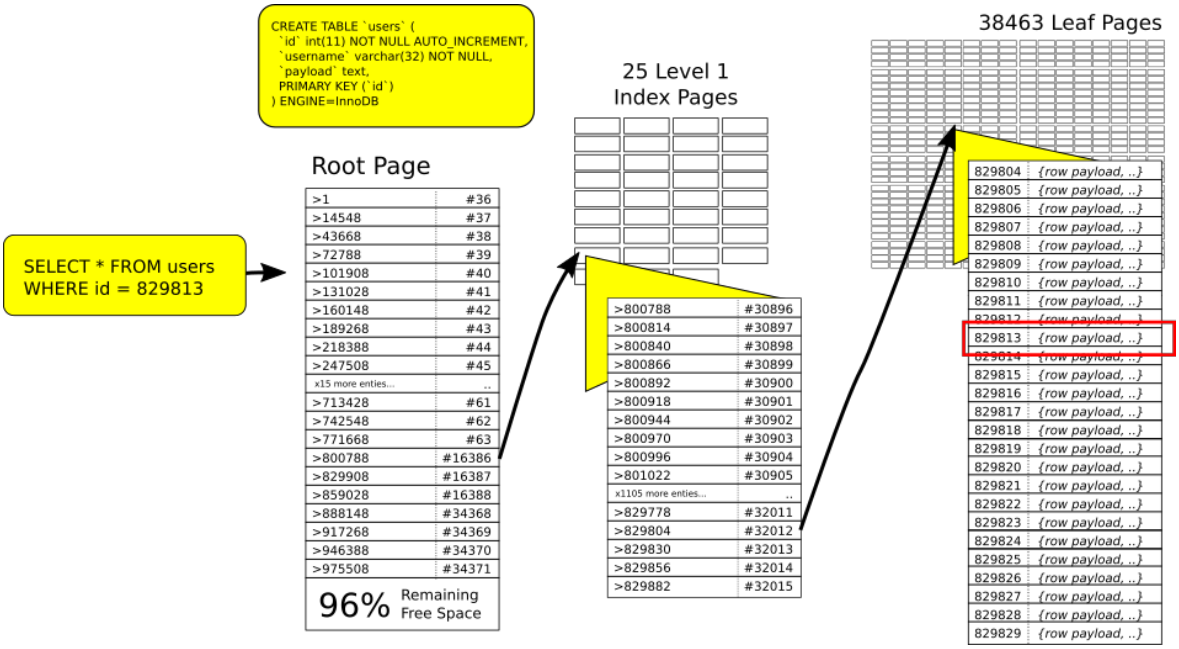
### *1.3.8 Indexovanie tabuliek v databáze*

Posledná možnosť konfigurácie databázových systémov sa netýkala priamo nastavení MySQL servera, ale obsahu tabuliek, resp. nastaveniu indexácie jednotlivých tabuliek v

databázovom systéme. Indexovanie stĺpcov môže zrýchliť načítanie výsledkov z tabuliek, avšak môže mať aj opačný dopad a teda spomaliť získavanie výsledkov.

Indexovanie v databáze funguje na princípe binárnych stromov. Binárne stromy sú rýchle a efektívne na získanie požadovaných výsledkov (Nichter, 2021). Samotné fungovanie binárnych stromov v tejto práci vysvetlené nie je, avšak podstatnou informáciou je, že v prípade pridania nového záznamu do tabuľky, ktorá obsahuje indexy je nutné binárny strom prepočítať. Práve toto prepočítanie môže spôsobovať komplikácie a spomalenie pri častom zápise do databázových systémov (Nichter, 2021).

*Obrázok 2: Binárny strom v MySQL databáze*




Zdroj: unofficialmysqlguide.com

Vzhľadom na tieto skutočnosti je pri návrhu tabuliek a indexov v databázovom systéme dôležité prihliadať aj na ich využitie. Ak ide o tabuľku do ktorej zapisujeme často ale údaje z nej získavame iba málokedy, napríklad tabuľka ukladajúca všetky zmeny vykonané v systéme, je možné indexovanie vynechať za účelom lepšieho výkonu databázového systému. Naopak ak hovoríme o tabuľke, ktorú nemeníme tak často – zoznam ponúkaných produktov, môžeme využiť indexovanie pre zrýchlenie načítania záznamov.

## 1.4 Technológie spojené s databázovými systémami

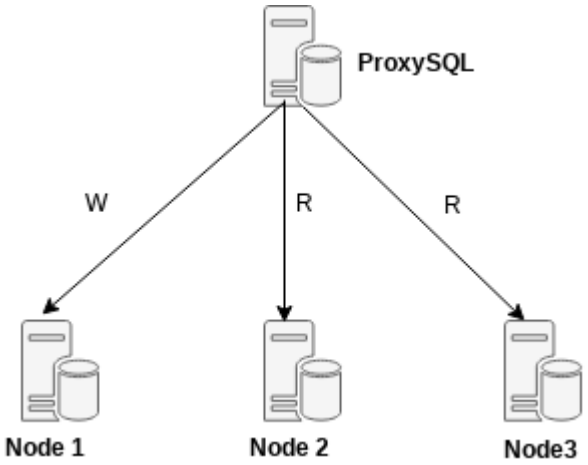
Pri návrhu distribuovaných databázových systémov sa okrem výberu vhodných technológii na prevádzku systému vieme zamerať aj na ďalšie súčasti, ktoré nám môžu

pomôcť optimalizovať systém. V tejto časti práce si predstavíme pár nástrojov, ktoré nám slúžia ako vstupný bod do databázového systému a slúžia na distribúciu záťaže medzi rôzne súčasti distribuovaného systému.

### *1.4.1 ProxySQL*

ProxySQL slúži ako jeden z nástrojov na distribúciu záťaže v distribuovanom systéme. Okrem toho, že poskytuje možnosť rozdistribuovať záťaž ponúka aj pokročilejšie funkcie ako kontrola stavu jednotlivých uzlov distribuovaného systému a v prípade komplikácii na jednom z uzlov dokáže presmerovať požadované dopyty na iné uzly v rámci systému (Risal, 2022). Ide o nástroj ktorý je stavaný primárne na distribúciu záťaže v databázových systémoch.

*Diagram 1: Architektúra ProxySQL*




Zdroj: mafiree.com

### *1.4.2 HaProxy*

Ako alternatívu *ProxySQL* je možné na rozloženie záťaže využiť nástroj *HaProxy*. Ide o reverzný proxy server zabezpečujúci vysokú dostupnosť a rozloženie záťaže pre HTTP a TCP spojenia (HAProxy, n. d.). Nejde o systém, ktorý by slúžil primárne pre databázové systémy, avšak vzhľadom na to, že databázy využívajú na komunikáciu TCP spojenia je možné využiť na distribúciu záťaže aj tento nástroj (Kanyi, 2018). Keďže slúži na distribúciu TCP komunikácie, neponúka pokročilú funkcionalitu pre prácu s databázami ako ProxySQL, avšak podľa návrhu infraštruktúry je možné ho využiť ako jednu reverznú proxy, ktorá nám v organizácii pokrýva viacero systémov.

## 1.5 Sieťové spojenie

Databázové systémy primárne komunikujú prostredníctvom sieťových spojení, či už je to v rámci distribuovaného databázového systému v internej sieti alebo komunikácia medzi koncovým užívateľom alebo analytikom a databázovým systémom. Vzhľadom na to si v tejto kapitole predstavíme niektoré pojmy, ktoré v sieťovej komunikácii môžu spôsobovať komplikácie.

### *1.5.1 Latencia*

Latenciu vieme opísať ako čas alebo oneskorenie v čase medzi tým kedy odošleme dáta po sieti a kedy sú prijaté. Meriame ju v milisekundách a za optimálnu odozvu môžeme považovať hodnoty v rozmedzí 30 až 40ms, zatiaľ čo latencia vyššia ako 100ms je považovaná za vysokú (IR Team, n. d.).

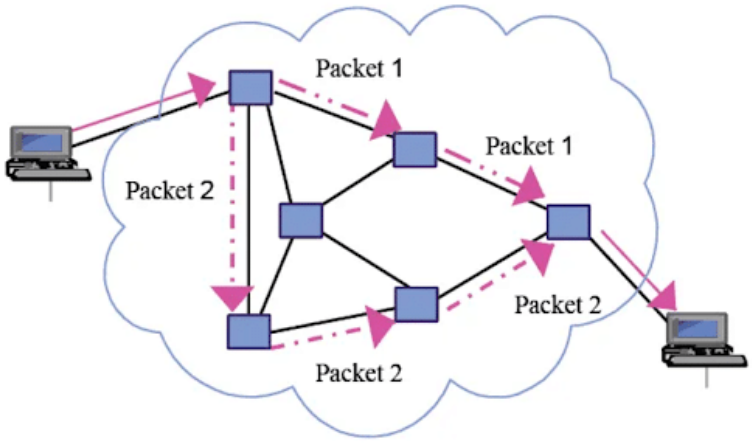
Latencia môže byť spôsobená viacerými faktormi. Jedným z nich je fyzická vzdialenosť medzi zariadeniami, ktoré spolu komunikujú. Platí, že čím väčšia vzdialenosť je medzi zariadeniami, tým bude väčšia latencia, vzhľadom na fyzikálne zákony. Okrem toho môže byť latencia spôsobená aj komplikáciami na sieti, prenosovým médiom (optický kábel, bezdrôtové spojenie) alebo preťažením zariadení (Goodwin, 2023).

Pri návrhu distribuovaného databázového systému chceme minimalizovať latenciu nakoľko môže mať negatívny vplyv na fungovanie celého nášho systému.

### *1.5.2 Strata paketov*

Pakety sú základnou dátovou jednotkou prenášanou prostredníctvom TCP/IP siete. Obsahujú dve kategórie dát – kontrolné informácie (zdrojová a cieľová sieťová adresa, sekvenčné informácie, chybové kódy) a používateľské dáta (obsah samotnej správy) (Drake, 2021).

*Obrázok 3: Cesta paketov medzi dvoma zariadeniami*




Zdroj: ringcentral.com

Pri internetovej komunikácii sú všetky správy rozdelené do menších celkov – paketov, ktoré následne ako samostatné jednotky cestujú po sieti do cieľovej destinácie. Rozdelenie správ na menšie časti pomáha zabrániť preťaženiu sieti (Jameson, 2024).

Strata paketov znamená, že nejaká časť odoslanej správy nedorazila do cieľovej destinácie. Meria sa v percentách a ak do cieľovej destinácie nedorazí jeden zo sto paketov, môžeme povedať, že ide o stratu paketov vo výške 1%. Dôvodov prečo sa tak môže stať je viacero, ako napríklad zahltená sieť, nesprávna konfigurácia sieťových zariadení alebo firewallu, DDoS útoky a podobne (PubNub Labs Team, 2024). V prípade, že príde ku strate paketov v TCP sieti, tento protokol je navrhnutý tak aby opäť odoslal stratený paket (Pandora FMS team, 2025).

## 1.6 Iné využívané technológie a pojmy

Okrem technológii a pojmov zameriavajúcich sa na distribuované databázové systémy sme sa v tejto práci mohli stretnúť aj s inými pojmami a technológiami, ktoré síce nesúvisia s databázovými systémami, ale naprieč optimalizáciou systému boli do veľkej miery využívané.

### *1.6.1 Proxmox Virtual Environment*

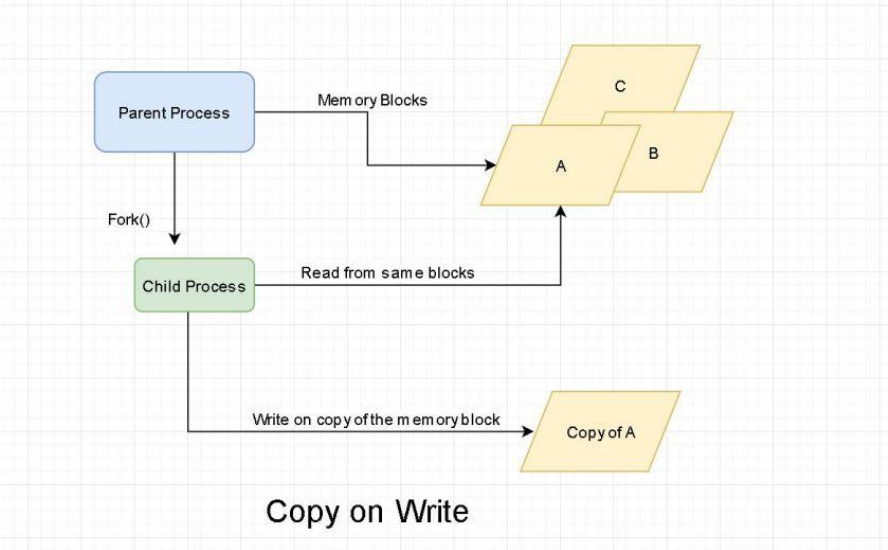
*Proxmox Virtual Environment* je operačný systém založený na *GNU/Linux* a operačnom systéme *Debian* (Proxmox Server Solutions GmbH., n. d.). Ide o operačný systém s otvoreným zdrojovým kódom a zameriava sa na poskytovanie nástrojov

umožňujúcich jednoduché vytváranie virtuálnych strojov. Považujeme ho za hypervisor typu 1, teda hypervisor, ktorý komunikuje priamo s hardvérom na ktorom je spustený. Pre porovnanie, hypervisor typu 2 využíva pre svoju funkčnosť funkcie operačného systému na ktorom je nainštalovaný. Za typ 2 hypervizor môžeme považovať napríklad *Oracle VM VirtualBox* na *OS Windows* (Simic, 2022).

### *1.6.2 ZFS*

*ZFS* je súborový systém založený na princípe *COW* (*Copy-on-write*) a okrem správy súborov ponúka možnosti  správy logických diskov a softvérový *RAID* (Wojsław, 2017). Benefitom *ZFS* je taktiež aj dočasné ukladanie súborov do pamäte RAM, tzv. *ARC cache*, ktorá zvyšuje rýchlosť operácii čítania a zápisu najnovších súborov a súborov ku súborom, ktoré sú najčastejšie využívané (Wojsław, 2017). Tento benefit môžeme využiť pri návrhu databázového systému, nakoľko môže tabuľky uložiť do systémovej pamäte a zrýchliť budúce dopyty.

*Obrázok 4: Copy-on-write*




Zdroj: learnsteps.com

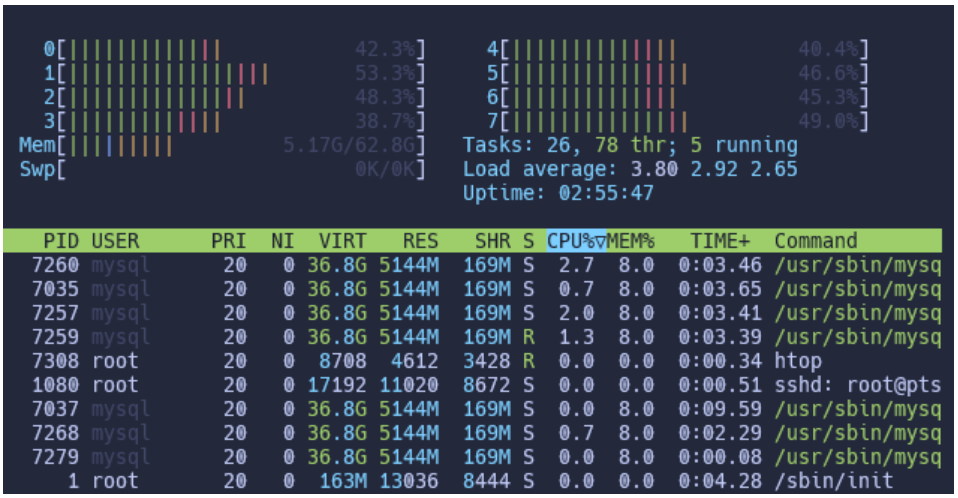
### *1.6.3 Rozdiel medzi zaťažením a využitím CPU*

V neskorších častiach tejto práce sa stretneme s pojmami zaťaženie CPU a využitie CPU. Ako už vyplýva z názvu, v oboch prípadoch hovoríme o metrike, ktorá meria aktivitu CPU na serveri.

Zaťaženie CPU je metrika, ktorú ponúkajú OS založené na *GNU/Linux* a vo väčšine prípadov sa skladá z troch hodnôt – zaťaženie za poslednú minútu, posledných päť minút a posledných pätnásť minút. Reprezentuje priemerný počet procesov, ktoré využívajú CPU alebo čakajú na pridelenie dostupných zdrojov CPU (Benny, 2017). Pre ilustráciu, ak máme zariadenie s dvoma jadrami a vidíme, že zaťaženie CPU je na úrovni tri, znamená to, že v priemere jeden proces čaká na pridelenie dostupných zdrojov a dva procesy aktívne využívajú CPU. Do tejto kalkulácie sa rátajú aj procesy, ktoré nemusia čakať výhradne na CPU ale môže ísť aj o procesy čakajúce na diskové polia – teda IO operácie (Tomer, 2024). Túto hodnotu je nutné interpretovať v kombinácii s počtom CPU jadier, ktoré ma systém k dispozícii (SentinelOne, 2021).

Využitie CPU sa meria v percentách a vyjadruje hodnotu času, kedy CPU vykonávalo nejakú úlohu. V ideálnej situácii pri serveroch chceme aby sa využitie približovalo maximu, ale zároveň aby neprekročilo hodnotu 80-90% (Benny, 2017). Je to z dôvodu, snahy optimalizovania využitia zdrojov s čo najlepším výsledkom (Tomer, 2024).

*Obrázok 5: Príkaz htop na Ubuntu zobrazujúci využitie CPU zdrojov*




Zdroj: vlastné spracovanie

# 2 Cieľ práce

Ako bolo naznačené v úvode tejto práce, rôzne spoločnosti a organizácie sa čoraz viac spoliehajú na databázové systémy na uchovávanie ich dát. Či už hovoríme o zozname zamestnancov a ich nároku na voľno alebo tabuľke dostupných skladových zásob, s najväčšou pravdepodobnosťou vo väčšine firiem tieto údaje nájdeme uchované v databázovom systéme. Keďže tieto dáta sú životne dôležité pre fungovanie spoločností, je potrebné aby boli vysoko dostupné a zároveň aj čo najprístupnejšie pre užívateľov databázy (oddelenie ľudských zdrojov, sklad, analytikov spoločnosti, atď.).

V ďalších kapitolách tejto práce sme sa vzhľadom na vyššie uvedené pozreli na to, ako efektívne navrhnúť distribuovaný databázový systém vytvorený na systéme *MySQL Galera Cluster*. Pri tomto systéme sme sa pozreli na návrh samotného systému, jeho rozloženia naprieč servermi a následnou optimalizáciou za účelom čo najväčšieho počtu spracovaných dopytov za určitý časový rámec.

Okrem podrobného pohľadu na *MySQL Galera Cluster* sme sa pozreli na tému optimalizácie distribuovaného databázového systému aj zo všeobecnejšieho pohľadu a zamerali sme sa na testovanie a optimalizáciu vplyvu sieťového spojenia medzi jednotlivými servermi v distribuovanom systéme. Taktiež sme sa pozreli aj na technológie mimo distribuovaný systém, ktoré nám mohli pomôcť s efektívnejším rozdelením záťaže na databázový systém – menovite *ProxySQL*.

Na záver sme sa vo výsledkoch práce zamerali na predstavenie návrhu distribuovaného databázového systému, ktorý zahŕňal sieťovú optimalizáciu a *ProxySQL* server na rozdelenie záťaže. Cieľom bolo upraviť konfiguráciu jednotlivých súčastí distribuovaného databázového systému tak aby pri využití rovnakých hardwarových zdrojov dokázal databázový systém spracovať čo väčšie množstvo dotazov v porovnaní so základnou konfiguráciou.

# 3 Metodika práce a metódy skúmania

V časti metodiky sme sa zamerali na podrobný opis infraštruktúry a nástrojov, ktoré sme počas testovania a optimalizácie distribuovaného databázového systému zvolili. Pozreli sme sa na fyzický hardvér, návrh distribúcie systému ktorý bol využívaný vo väčšine vykonávaných testov ale zamerali sme sa aj na priblíženie testovacieho nástroja a jeho výstupu. Na záver tejto kapitoly sme si vysvetlili metodiku výpočtu výsledkov testov, ktorá slúžila ako kritérium optimálnosti optimalizácie distribuovaného databázového systému.

## 3.1 Parametre testovacích serverov

Pre vykonávanie testov a optimalizácie distribuovaného databázového systému sme mali k dispozícii jeden server typu *Dell PowerEdge R640*. Tento server mal nasledovné parametre:

- Procesor: *2x Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz*
- RAM: *256 GB*
- OS: *Proxmox Virtual Environment 8.3.3*
- Verzia Linux kernelu: *Linux 6.8.12-7-pve*

Vzhľadom na to, že na testovanie sme mali dostupný iba jeden server, rozhodli sme sa testovanie optimalizácie vykonávať formou vytvorenia niekoľkých virtuálnych strojov, ktoré slúžili ako náhrada za fyzické stroje. Pri virtuálnych strojoch sme sa čo najviac snažili oddeliť disky, ktoré mali k dispozícii aby neprichádzalo ku obmedzovaniu výkonu na základe dlhej odozvy diskov, tzv. vysokého IO wait. Zároveň na tomto serveri neboli prevádzkované žiadne iné služby okrem súčastí distribuovaného databázového systému a zabudovaných monitorovacích nástrojov. Týmto krokom sme sa snažili čo najviac priblížiť skutočnosti, v ktorej spoločnosti majú dedikované servery iba pre spracovanie náročných databázových dopytov a ostatné služby sa nachádzajú na iných serveroch.

Disky ktoré sme využívali boli typu SATA SSD a využívali súborový systém *ZFS*. Tento súborový systém nám umožnil disky spojiť do diskových polí, čím sa zvýšila redundancia úložného priestoru. Okrem toho sme sa pre tento súborový systém rozhodli z dôvodu možnosti vytvárania tzv. snapshotov jednotlivých častí diskových polí. Tieto snapshoty sú rýchle na vytvorenie a je možné ich odoslať na iné zariadenie, napríklad na server, ktorý slúži iba na zálohovanie dát. Teda poskytujú nám ďalšiu úroveň zálohovania systémov. Táto

vlastnosť bola pre nás podstatná vzhľadom na rôzne testy, ktoré sme vykonávali a zabezpečila, že virtuálne stroje nebolo nutné preinštalovať po každom vykonanom teste.

Jedna z funkcií *ZFS* je taktiež aj ukladanie nedávno otvorených a najčastejšie otvorených súborov do pamäte RAM (45 Drives Ltd., n. d.). Táto vlastnosť môže v niektorých situáciách zrýchliť dopytovanie sa na databázu. Avšak optimalizácia a testovanie vplyvu *ZFS* na chod databázového systému je nad rámec tejto práce.

## 3.2 Operačný systém

Ako sme spomenuli v predchádzajúcich častiach tejto práce, na návrh distribuovaného databázového systému sme mali k dispozícii iba jeden fyzický server. Z toho dôvodu sme sa rozhodli využiť virtualizáciu. Potreba virtualizácie nás priviedla ku voľbe operačného systému *Proxmox Virtual Environment*. Ide o operačný systém využívajúci *Linux* kernel, založený na *OS Debian GNU/Linux* s verejne dostupným zdrojovým kódom (Proxmox Server Solutions GmbH., n. d.). V porovnaní s inými OS ako napr. *VMWare ESXi* ide o bezplatný operačný systém, ktorý ponúka jednoduché a intuitívne grafické rozhranie na vytváranie a manažment virtuálnych strojov. Druhým benefitom je natívna podpora *ZFS* súborového systému.

Okrem týchto vlastností ponúka *Proxmox Virtual Environment* aj iné benefity ako možnosť tvorby distribuovaného vysoko dostupného virtualizačného prostredia alebo sieťový súborový systém *Ceph*. Avšak tieto funkcie neboli pre nás v rámci obmedzení pri písaní práce rozhodujúcimi faktormi.

## 3.3 Databázový systém

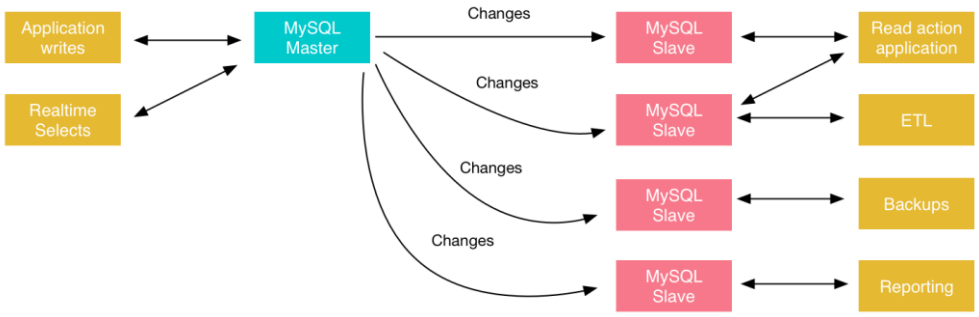
Pri návrhu distribuovaného databázového systému využívajúceho databázový systém *MySQL* sme mali na výber viacero možností distribuovaných systémov. Pred začiatkom návrhu bolo nutné zvoliť vhodný systém distribúcie dát. V nasledujúcich podkapitolách si predstavíme rôzne spôsoby dosiahnutia distribuovaného databázového systému na báze *MySQL* spolu s ich benefitmi a na záver tejto kapitoly sa pozrieme na dôvody, pre ktoré sme si zvolili našu implementáciu distribuovaného databázového systému.

### *3.3.1 MySQL Replication*

Za najzakladanejšiu formu *MySQL* distribuovaného systému môžeme považovať replikáciu dát, ktorá je zabudovaná v *MySQL* serveri bez nutnosti dodatočnej inštalácie iných distribučných nástrojov. Replikácia funguje na princípe jedného hlavného *MySQL* servera a

jednej alebo viacero replík na ktoré sú kopírované dáta z hlavného *MySQL* servera. Replikácia je iba jednosmerná, teda v prípade jej využitia môžeme vykonávať zápis a úpravu dát iba na hlavnom *MySQL* serveri (čítania je možné vykonávať aj z replík) (Oracle Corporation, n. d.)(a). Replikácie je zároveň asynchrónna, teda nie je nutné aby repliky mali sieťové spojenie s hlavnou databázou po celú dobu ich existencie.

*Diagram 2: MySQL replikácia*




Zdroj: percona.com

Pri využití tohto spôsobu máme k dispozícii dve možnosti replikácie dát. Jedna z nich je na základe tzv. binárnych logov (*binary log*) a druhá na základe *GTID – global transaction identifiers* (Oracle Corporation, n. d.)(a). Rozdiel v týchto spôsoboch sme si predstavili v prvej kapitole tejto práce.

Výhodou *MySQL* replikácie je možnosť odbremenenia hlavnej databázy od operácii čítania. Taktiež je možné repliky využiť ako formu záloh alebo v prípade nutnosti zdieľania dát s tretími stranami, kedy im nemusíme zdieľať prístup do hlavnej databázy, ale je možné vytvoriť repliku, ktorá obsahuje iba požadované tabuľky.

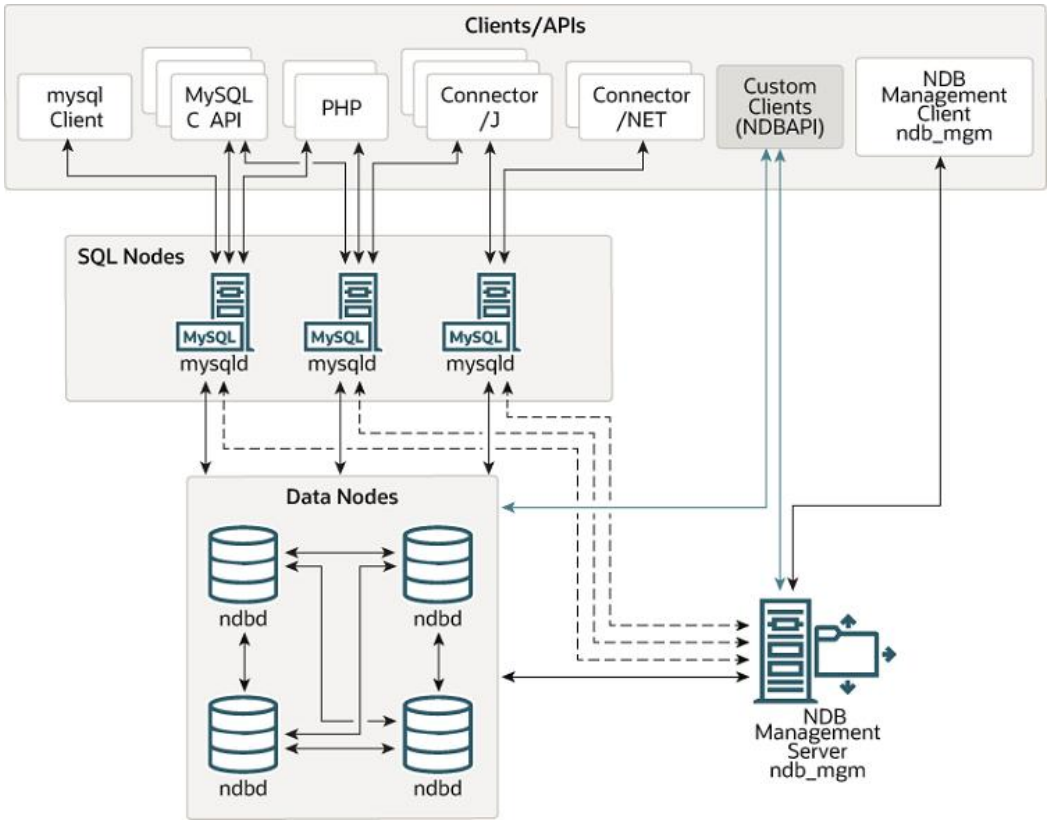
Za nevýhody môžeme považovať možnosť vykonávania zápisu iba na jednu, hlavnú, databázu, čo v systéme ktorý by mal byť vysoko dostupný predstavuje miesto pri ktorom návrh stráca vysokú dostupnosť. Vzhľadom na vyššie uvedené sme nepovažovali *MySQL* replikáciu za vhodný nástroj na implementáciu distribuovaného databázového systému.

### *3.3.2 MySQL NDB Cluster*

Ako druhú možnosť na vytvorenie distribuovaného systému máme nástroj *MySQL NDB Cluster*. Ide o externé rozšírenie *MySQL*, ktoré je vyvíjané spoločnosťou Oracle. Je tvorený z troch serverových súčastí a to SQL uzlov, dátových uzlov a manažment servera a umožňuje prevádzkovať viac ako dvesto *MySQL* serverov, ktoré ukladajú rovnaké dáta

(Blåudd, 2024). Tieto servery majú architektúru *active-active*, teda zápis aj čítanie dát je možné z ktoréhokoľvek servera v rámci distribuovaného systému.

*Diagram 3: MySQL NDB cluster*




Zdroj: dev.mysql.com

Oproti *MySQL* replikácii, alebo *MySQL Galera Clusteru*, ktorý si priblížime v ďalšej časti tejto práce, využíva na pozadí vlastnú úložnú technológiu s názvom *NDB – Network Database*. Ide o úložnú technológiu, ktorá neukladá údaje na disk ako pri *MyISAM* alebo *InnoDB* (úložné technológie bežne využívané v *MySQL*) ale primárne ukladá všetky dáta do pamäte RAM (Davies, a iní, 2006).

Za výhodu môžeme považovať ukladanie dát do pamäte RAM, čo zvyšuje rýchlosť čítania a zápisu dát do databázového systému. Avšak v niektorých situáciách môže byť zapisovanie do pamäte RAM nevýhodou. Taktiež ak v databáze plánujeme využívať cudzie kľúče, môžeme pozorovať dopad na výkon (Oracle Corporation, n. d.)(b).

Vzhľadom na to, že tento distribuovaný systém je stavaný primárne na zápis do pamäte RAM, pomerne nízke množstvo dokumentácie (okrem oficiálnej dokumentácie od spoločnosti Oracle) a náročnosť konfigurácie pre základný chod distribuovanej databázy

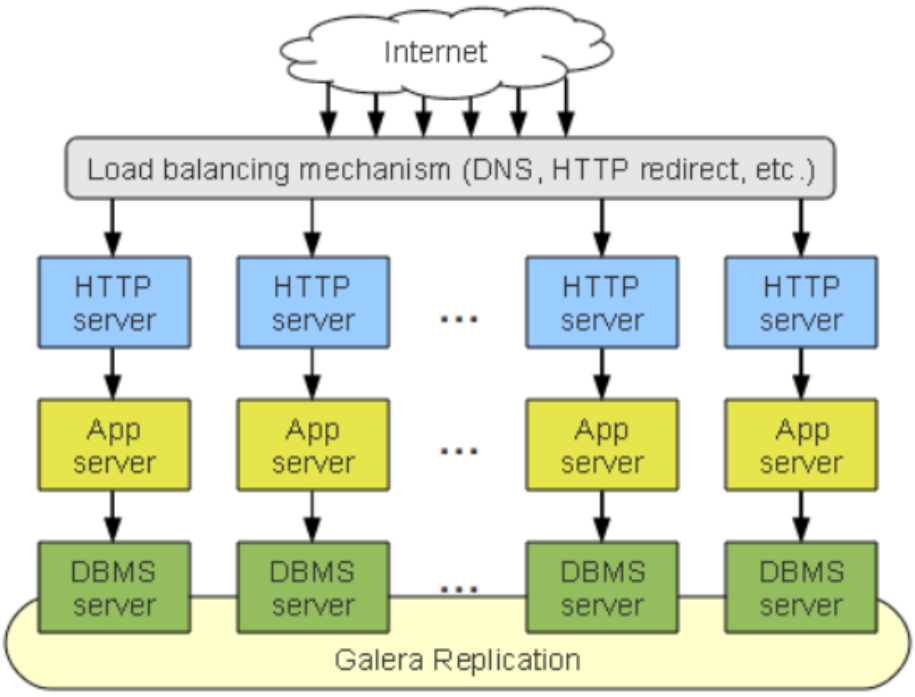
sme nepovažovali *MySQL NDB Cluster* za vhodnú alternatívu pre distribuovaný databázový systém, ktorý sme navrhovali.

### *3.3.3 MySQL Galera Cluster*

Tretí distribuovaný databázový systém ktorý sme v rámci návrhu zvažovali bol *MySQL Galera Cluster*. Tento systém je vyvíjaný spoločnosťou *Codership Oy*. Ide o systém, ktorý umožňuje vytvorenie distribuovaného databázového systému s viacerými primárnymi servermi.

Na pozadí využíva vlastnú *Write-Set Replication API* a dáta ukladá za pomoci úložného systému *InnoDB*, ktorý je využívaný aj mimo distribuované databázové systémy. Keďže *InnoDB* ukladá dáta na disky, aj v prípade, že by nastala komplikácia kvôli ktorej by sa reštartovali všetky uzly distribuovaného systému, nepríde ku strate dát (Codership Oy, 2025).

*Diagram 4: MySQL Galera Cluster*




Zdroj: galeracluster.com

Nakoľko *Galera Cluster* využíva na pozadí rovnakú úložnú technológiu ako *MySQL*, pri návrhu distribuovaného databázového systému je čiastočne možné aplikovať znalosti z optimalizácie databázových serverov.

Za nevýhody *Galera Clusteru* môžeme považovať jeho dostupnosť iba na operačných systémoch založených na Linuxe, nutnosť mať v tabuľkách primárny kľúč a pre niektorých databázových administrátorov môže byť nevýhodou aj to, že tento systém je navrhnutý primárne na úložnom systéme *InnoDB* (Razzoli, 2014).

Ako benefit môžeme považovať synchrónnu replikáciu, ktorá garantuje rovnaké dáta naprieč všetkými uzlami v distribuovanom clusteri, možnosť zapisovať na ktorýkoľvek uzol, čo umožňuje distribúciu záťaže naprieč viacerými systémami alebo podobnosť nastavení so základnou verziou *MySQL*, čo do značnej miery redukuje náročnosť údržby tohto distribuovaného databázového systému.

Keďže tento distribuovaný databázový systém ponúka možnosť uloženia dát na disk, synchrónnu replikáciu, možnosť zápisu na všetky uzly a podobnosť konfigurácie s bežným *MySQL* serverom, v tejto práci sme sa rozhodli, že náš distribuovaný databázový systém bude fungovať na *MySQL Galera Cluster* technológii.

## 3.4 Distribúcia záťaže

Po výbere vhodného distribuovaného databázového systému sme sa pozreli na možnosti, akými vieme rovnomerne distribuovať záťaž na databázový systém. Pri výbere nástroja sme mali na výber medzi troma populárnymi nástrojmi – *HaProxy*, *ProxySQL* a *MySQL Router*. Dané nástroje sme podrobnejšie opísali v prvej časti tejto práce.

Vzhľadom na negatívne výsledky verejne dostupných testov *MySQL Routera* (Tusa, 2024) (Tusa, 2023) sme sa rozhodli tento nástroj nebrať do úvahy pri rozhodovaní sa. Vzhľadom na to nám ostali na výber nástroje *HaProxy* a *ProxySQL*.

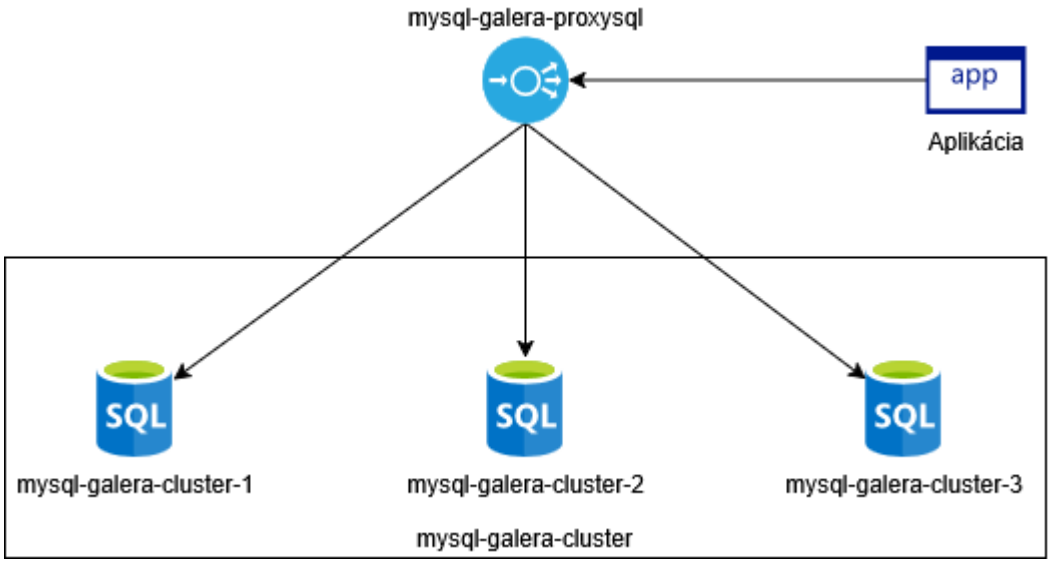
Nakoľko z nášho testovania, ktoré bližšie opisujeme v časti výsledkov práce, bol výkon *ProxySQL* a *HaProxy* takmer rovnaký, rozhodovali sme sa na inom kritériu. Kritériom pre nás bola dostupná funkcionalita a vzhľadom na to, že *ProxySQL* je nástroj, ktorý umožňuje hlbšiu konfiguráciu distribúcie záťaže pre databázové systémy, rozhodli sme sa ho zakomponovať v našom návrhu distribuovaného databázového systému.

## 3.5 Návrh distribuovaného systému

V tejto práci sme testovali viacero systémov distribuovaného databázového systému, ktorých pozitíva a negatíva sme opisovali v predchádzajúcich častiach tejto práce. Po zvolení vhodného systému bolo nutné navrhnúť vhodnú architektúru. Pri návrhu architektúry

distribuovaného databázového systému sme brali do úvahy požiadavky *MySQL Galera Clusteru*, ktorý pre vysokú dostupnosť požaduje minimálne tri servery (Razzoli, 2014). Okrem toho sme mali záujem využiť aj možnosť zápisu na všetky uzly. Vzhľadom na to sme sa rozhodli pre distribuovaný systém skladajúci sa zo 4 serverov – jeden server slúžil ako proxy server využívajúci nástroj *ProxySQL* a na zvyšných troch serveroch sa nachádzali SQL servery typu *MySQL Galera Cluster*. Externé aplikácie, v našom prípade testovací nástroj, komunikovali výhradne s *ProxySQL* serverom, ktorý následne dopyty preposielal na jednu z troch replík v rámci *MySQL Galera Clusteru*. Pre lepšiu ilustráciu, nižšie uvádzame aj diagram návrhu systému:

*Diagram 5: Návrh distribuovaného databázového systému*




Zdroj: Vlastné spracovanie

Všetky súčasti distribuovaného systému sa nachádzali na individuálnych virtuálnych strojoch, ktoré využívali *OS Ubuntu 24.04.5 LTS* s kernelovou verziou *GNU/Linux 5.15.0-133-generic x86_64*.

## 3.6 Testovanie sieťového spojenia

Nakoľko sme sa v práci zameriavali na celkovú optimalizáciu distribuovaného databázového systému, ktorá obsahovala aj testovanie vplyvu sieťového spojenia na chod distribuovaného databázového systému, avšak mali sme k dispozícii iba jeden fyzický stroj na testovanie, bolo potrebné do testov zahrnúť aj umelé komplikácie spojené so sieťou.

Pre tento účel sme si vybrali sadu nástrojov *NetEm* a *tc*. Ide o súčasť *Linuxu*, v ktorom je typicky súčasťou balíčku *iproute2* a umožňuje pridanie umelých komplikácii so sieťovým spojením ako napríklad pridanie odozvy, straty paketov a ďalších na odchádzajúce spojenia (SRTLab, 2023). *NetEm* je založený na už existujúcej funkcionalite kernelu Linuxu, ktorú využíva ku dosiahnutiu požadovaných výsledkov (Free Software Foundation, 2011).

Tento nástroj sme nasadzovali na *ProxySQL* server, ktorý slúžil ako vstupný bod pre testovací nástroj, resp. pre akúkoľvek aplikáciu, ktorá by ukladala dáta do databázového systému.

## 3.7 Testovací nástroj DBT2

Pre testovanie výkonu distribuovaného databázového systému sme sa rozhodli využiť nástroj *DBT-2 – OSDL Database Test 2*. Tento test je derivovaný z testovacej špecifikácie zverejnenej *Transaction Processing Performance Council (TPC)* (Open Source Development Labs, Inc., n. d.).

TPC-C špecifikácia sa zameriava na reprezentáciu databázovej aktivity v spoločnosti, ktoré manažuje, predáva a distribuuje produkty alebo služby. Testovanie simuluje prostredie v ktorom operátori vykonávajú rôzne transakcie voči databáze. Tieto transakcie obsahujú vkladanie a dodávanie objednávok, zadávanie platieb, kontrolu stavu objednávok a monitorovanie skladových zásob (Raab, a iní, n. d.).

### *3.7.1 Parametre testovania*

V rámci testovania sme využívali predvolené hodnoty testovacieho nástroja, ktoré boli nasledovné: 30 skladov, 10 územných celkov, 3000 zákazníkov, 100000 produktov, z ktorých mal každý z produktov 100000 skladových zásob, 3000 objednávok a 900 nových objednávok (Open Source Development Labs, Inc., n. d.). Celkové trvanie testu bolo nastavené na 5 minút a z každého skladu sa pripájalo ku databázovému systému 10 terminálov. Teda celkovo bolo vytvorených 300 spojení na databázový systém.

Testovací nástroj *DBT-2* sme zároveň prevádzkovali na inom zariadení, ktoré sa po lokálnej sieti s priepustnosťou 2,5Gbps pripájalo ku cieľovej databáze. Išlo o jeden z viacerých krokov, ktoré sme sa rozhodli implementovať do metodiky testovania za účelom čo najpresnejšieho odzrkadlenia reality pri dostupných zdrojoch.

Nástroj vykonával testy v piatich kategóriách databázových transakcií:

### New-Order Transaction

Test nových objednávok vykonával čítanie a zápis do databázy. Každá transakcia obsahovala sedem až sedemnásť výberov riadkov, šesť až šestnásť výberov riadkov spolu s ich aktualizáciou a sedem až sedemnásť pridaných riadkov. Tento test bol vykonávaný približne 45% času (Open Source Development Labs, Inc., n. d.).

### Payment Transaction

Test platobných transakcií bol nenáročným testom, testujúcim čítanie a zápis do databázy, ktorý aktualizoval stav účtov zákazníkov a odzrkadľoval platby v štatistikách územných celkov a skladov. Táto transakcia vykonávala v priemere dva výbery, šesť výberov s aktualizáciou údajov a dva vklady do databázy. Test bol vykonávaný približne 43% času (Open Source Development Labs, Inc., n. d.).

### Order-Status Transaction

Test kontroly stavu objednávok bol priemerne náročnou operáciou nad databázou, ktorá vykonávala iba čítanie nedávnych objednávok zákazníkov. Test vykonával dva výbery riadkov a deväť až devätnásť výberov riadkov spolu s ich aktualizáciou. Test bol vykonávaný približne 4% času (Open Source Development Labs, Inc., n. d.).

### Delivery Transaction

Test dodávok spracovával do desať nových objednávok a vykonával výber dvoch riadkov, šesť až šestnásť výberov riadkov spolu s ich aktualizáciou a zmazanie jedného riadku. Bol vykonávaný približne 4% času (Open Source Development Labs, Inc., n. d.).

### Stock-Level Transaction

Test vykonával kontrolu dostupných skladových zásob. Išlo o test, ktorý bol náročný na operácie čítania z databázy a kontroloval počet nedávno predaných produktov, ktoré mali skladové zásoby nižšie ako stanovený limit. Tento test vykonával do 900 výberov riadkov približne 4% času.

### *3.7.2 Výstup testu*

Po dokončení testu nám testovací nástroj *DBT-2* poskytol nasledujúci výstup:

```
                     Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   4.03      2.349 :     3.071         1682                0   0.00
   New Order  44.62      2.094 :     2.825        18604              172   0.92
Order Status   3.84      2.070 :     2.824         1601                0   0.00
     Payment  42.80      2.209 :     2.967        17847                0   0.00
 Stock Level   4.00      2.299 :     2.986         1668                0   0.00

3698.73 new-order transactions per minute (NOTPM)
5.0 minute duration
0 total unknown errors
172.00 rollback transactions
1 second(s) ramping up
```

V tomto výstupe môžeme vidieť výsledky jednotlivých kategórii transakcií, ich percento vykonávania v rámci testu, dĺžku testu, odozvu transakcií, počet vykonaných dopytov v jednotlivých kategóriách transakcií a celkové štatistiky výsledku testu.

Keďže v rámci tejto práce sa snažíme optimalizovať databázu celkovo, hodnota NOTPM – počet nových objednávkových transakcií za minútu pre nás nie je úplne relevantná. Výpočet, ktorým sme merali optimalizáciu a rýchlosť databázového systému sme si predstavili v neskoršej podkapitole tejto práce.

## 3.8 Výpočet rýchlosti databázového systému

Aby sme vedeli kvantifikovať vplyv zmien v databázovom systéme na jeho rýchlosť bolo nutné si pripraviť vhodný výpočet. Ako bolo spomenuté v predchádzajúcej podkapitole, testovací nástroj DBT2, ktorý využívame, ponúka vo výstupe ako primárnu metriku počet nových objednávok za minútu. Táto metrika sa nám zdala nedostatočná, keďže nebrala do úvahy ostatné operácie v rámci databázy. Vzhľadom na to sme sa rozhodli vytvoriť vlastný výpočet na kvantifikáciu vplyvu zmien. Pre porovnanie výkonu sme využívali vážený priemer s nasledovným vzorcom:

$$avg = \frac{(d * 4) + (no * 45) + (os * 4) + (p * 43) + (sl * 4)}{4 + 45 + 4 + 43 + 4} / 5$$

kde:

- $d$ = *Delivery Transactions* – počet spracovaných dodávok
- $no$ = *New-Order Transactions* – počet nových objednávok
- $os$ = *Order-Status Transactions* – počet dopytov na zistenie stavu objednávky

- $p$ = *Payment Transactions* – počet transakcií zameraných na tabuľku platieb
- $sl$ = *Stock-Level Transactions* – počet transakcií kontrolujúcich skladové zásoby

Váhy váženého priemeru jednotlivých hodnôt sme stanovili podľa percenta vykonávania času jednotlivých operácii. Na konci sme celý priemer vydelili hodnotou 5, ktorá predstavovala dĺžku trvania testu. Ako výsledku hodnotu sme dostali vážený priemer vykonaných SQL transakcií nad databázou za jednu minútu.

## 3.9 Monitorovanie využitia dostupných zdrojov

Počas vykonávania jednotlivých testov sme monitorovali aj záťaž na hardvér, na ktorom sa databázové servery nachádzali. Rozhodli sme sa merať štyri hodnoty a to – percento využitia CPU, priemernú záťaž CPU za poslednú minútu, čakanie na IO operácie (tzv. IO wait) a využitie CPU jednotlivých virtuálnych strojov nachádzajúcich sa na serveri. Operačnú pamäť RAM sme sa rozhodli po prvých testoch nezahrnúť do výsledkov tejto práce, nakoľko sme zistili, že zmeny v nastaveniach mali zanedbateľný vplyv na využitie pamäte RAM.

Pre monitorovanie sme využívali zabudovaný monitorovací server v *OS Proxmox*, ktorý nám umožňoval odosielať rôzne štatistiky o využití servera do *InfluxDB* databázového systému. Databázový systém *InfluxDB* sme umiestili mimo náš testovací server aby sme obmedzili jeho dopad na rýchlosť testovaného databázového systému a v konečnom dôsledku na výsledky záverov tejto práce.

Pre vizualizáciu výsledkov sme využili nástroj s otvoreným zdrojovým kódom – *Grafana*, ktorý ponúkal zabudované prepojenie s *InfluxDB* databázovým systémom. Tento nástroj nám ponúkol jednoduché prepojenie s databázou a následne umožnil vizualizovať rôzne štatistiky servera. Avšak v rámci tejto práce sme sa rozhodli neuvádzať obrázky a výstupy z tohto nástroja. Keďže ponúkal exportovanie výstupov vo formáte *.csv*, dáta sme mohli importovať do nástroja *MS Excel* a výstupné dáta mohli byť prezentované priamo v tabuľkách a grafoch natívnych pre *MS Word*.

# 4 Výsledky práce a diskusia

V časti výsledky práce a diskusia sme sa pozreli na kroky, ktoré boli vykonávane za účelom optimalizácie distribuovaného databázového systému. Následne boli tieto zmeny vyhodnotené na základe metodiky opísanej v predchádzajúcej časti práce. V prípade pozitívneho dopadu na databázový systém boli úpravy ponechané ako súčasť výsledného produktu a nastal prechod do ďalšieho kroku optimalizácie systému.

## 4.1 Testovanie softvérových a sieťových súčastí

Než sme sa zamerali na optimalizáciu samotných nastavení databázového systému, bolo nutné získať počiatočné hodnoty s ktorými budeme porovnávať výsledky. Tieto výsledky sme získavali naprieč touto kapitolou vykonávaním rôznych testov až sme našli optimálne riešenie softvérových balíčkov ale aj sieťových nastavení a nastavení indexácie v databáze.

### *4.1.1 Počiatočné testovanie*

Na úvod sme sa pozreli na výkon nedistribuovanej verzie databáz *MySQL* a *MariaDB*. *MySQL* aj *MariaDB* boli testované na najnovších verziách dostupných pre operačný systém. Pre *MySQL* išlo o verziu 8.0 a v prípade *MariaDB 10.5*. Hodnoty ktoré sme dostali slúžili ako základ pre naše ďalšie pozorovania a pokusy o optimalizáciu.

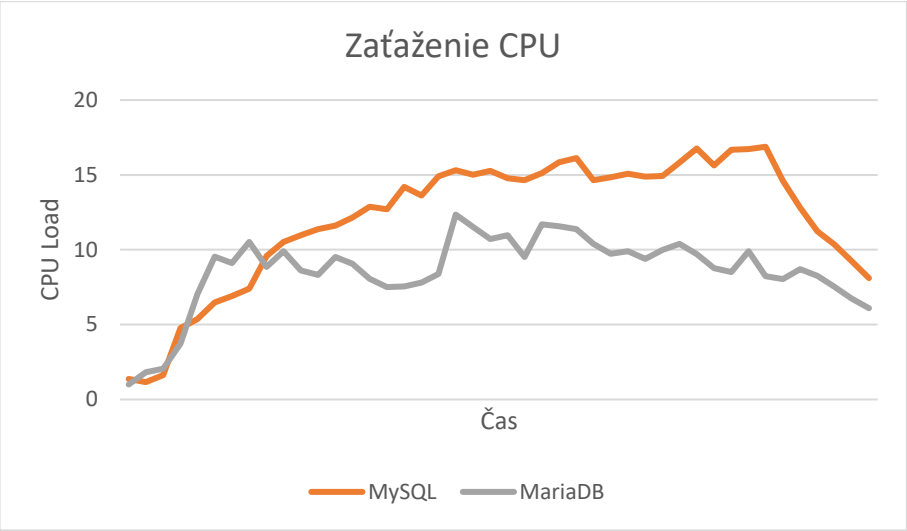
Testovanie prebiehalo na virtuálnych serveroch ktoré mali k dispozícii 32 jadier a 246GB RAM. Počas testov na fyzickom serveri neboli vykonávané žiadne iné operácie (okrem už spomenutého monitorovania) aby neprišlo ku skresleniu výsledkov testov.

*Tabuľka 1: Porovnanie výkonu MariaDB a MySQL*

| *Druh testu* | *MariaDB* | *MySQL* |
|---|---|---|
| *Delivery* | 2144 | 1682 |
| *New Order* | 24230 | 18604 |
| *Order Status* | 2151 | 1601 |
| *Payment* | 23223 | 17847 |
| *Stock Level* | 2182 | 1668 |
| *Vážený priemer* | *4229,69* | *3248,8* |

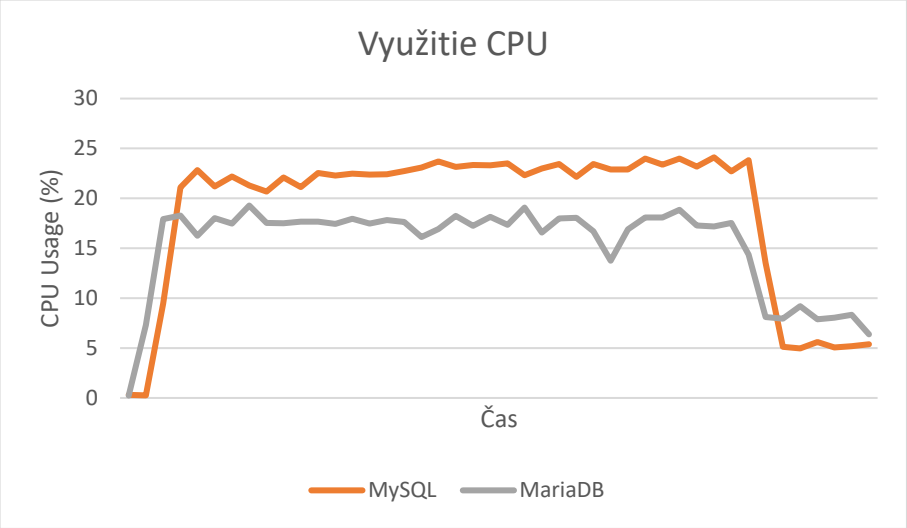
Zdroj: vlastné spracovanie

*Graf 1: Zaťaženie CPU pri porovnaní MySQL a MariaDB*



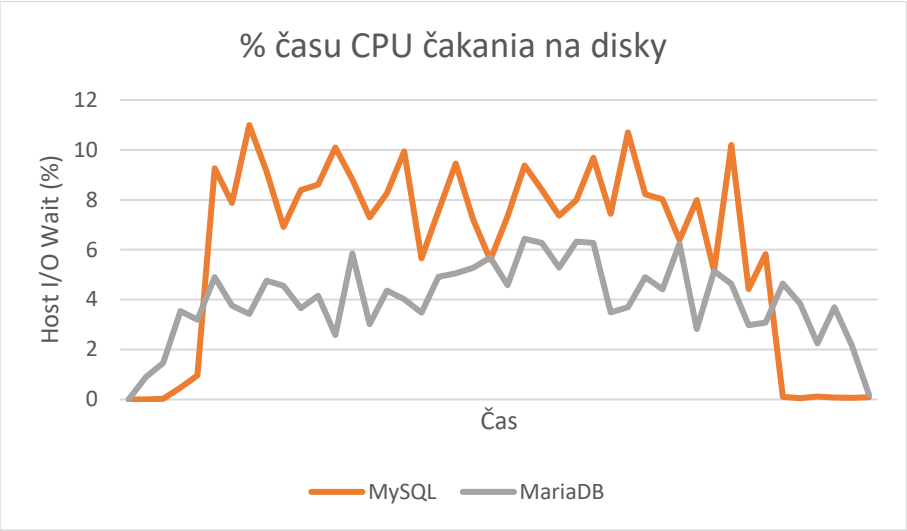

Zdroj: vlastné spracovanie

*Graf 2: Využitie CPU pri porovnaní MySQL a MariaDB*




Zdroj: vlastné spracovanie

*Graf 3: Percento času čakania na disky pri porovnaní MySQL a MariaDB*




Zdroj: vlastné spracovanie

Ako sme mohli vidieť z počiatočného testovania, databázový systém *MariaDB* dosahoval o 26% lepšie výsledky bez úpravy nastavení konfiguračných súborov v porovnaní s *MySQL*. Zároveň dokázal tieto výsledky dosiahnuť za využitia menších systémových zdrojov. Dosiahnuté hodnoty váženého priemeru boli v nasledovných častiach práce použité ako základ pre porovnania s distribuovaným databázovým systémom.

### *4.1.2 Vytvorenie distribuovaného databázového systému*

Po získaní všeobecných hodnôt sme vytvorili distribuovaný databázový systém na *MySQL Galera Cluster* a pozreli sme sa na výkonnostný dopad tohto systému. V tejto fáze nebol *Galera Cluster* optimalizovaný a išlo o predvolenú inštaláciu pri ktorej sme nakonfigurovali iba prepojenie medzi uzlami za využitia konfiguračného súboru uvedeného nižšie:

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="test_cluster"
wsrep_cluster_address="gcomm://10.10.0.58,10.10.0.74,10.10.0.24"
wsrep_sst_method=rsync
wsrep_node_address="XX.XX.XX.XX"
wsrep_node_name="mysql-galera-X"
```

Premenná *binlog_format* nám určuje, aký formát binárnych logov má byť využitý. Pre *MySQL Galera Cluster* je nutné mať tento formát typu *ROW*. *default-storage-engine* definuje predvolený úložný typ tabuliek databázy. V našom prípade išlo o formát *InnoDB*. *innodb_autoinc_lock_mode* s hodnotou „2“ umožňuje vykonávanie úloh v rámci systému na viacerých vláknach, teda paralelne. Premenná *bind-address* hovorí o IP adrese, na ktorej má databázový server počúvať. Hodnota *0.0.0.0* predstavuje všetky IP adresy, ktoré ma server priradené. *wsrep_on* s hodnotou „ON“ aktivuje replikáciu dát na ďalšie uzly v systéme. Toto nastavenie je nutné pre funkčnosť distribuovaného systému. *wsrep_provider* špecifikuje cestu v súborovom systéme ku knižnici *wsrep*. Bez tohto nastavenia, resp. pri nesprávnej ceste nie je možné používať distribúciu dát. Cesta ku knižnici môže byť odlišná podľa využívanej distribúcie *OS Linux*. *wsrep_cluster_name* a *wsrep_node_name* nám definujú názov distribuovaného systému a názov uzla. *wsrep_cluter_address* predstavuje zoznam IP adries všetkých uzlov, ktoré sa v rámci distribuovaného systému nachádzajú. Obsahuje aj IP adresu uzla na ktorom nastavujeme tento konfiguračný súbor. *wsrep_node_address* nadobúda hodnotu IP adresy uzla systému. Poslednou premennou bolo *wsrep_sst_method*, ide o premennú definujúcu spôsob, ktorým majú byť nakopírované dáta z jedného uzla na druhý v prípade inicializácie nového uzla do systému.

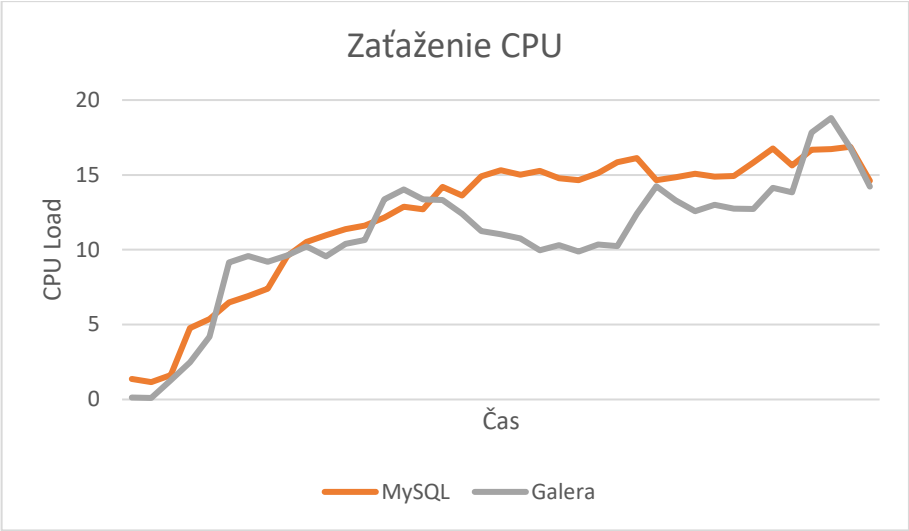
Testovanie prebiehalo na troch virtuálnych strojoch, každý z nich mal parametre 64GB RAM a 8 CPU jadier. Okrem MySQL virtuálnych strojov a monitorovacích nástrojov na fyzickom serveri sa na serveri nevykonávali žiadne iné operácie.

*Tabuľka 2: Porovnanie MySQL a MySQL Galera Cluster*

| *Druh testu* | *MySQL* | *Galera* |
|---|---|---|
| *Delivery* | 1682 | 1642 |
| *New Order* | 18604 | 18983 |
| *Order Status* | 1601 | 1628 |
| *Payment* | 17847 | 17947 |
| *Stock Level* | 1668 | 1712 |
| *Vážený priemer* | *3248,8* | *3291,768* |

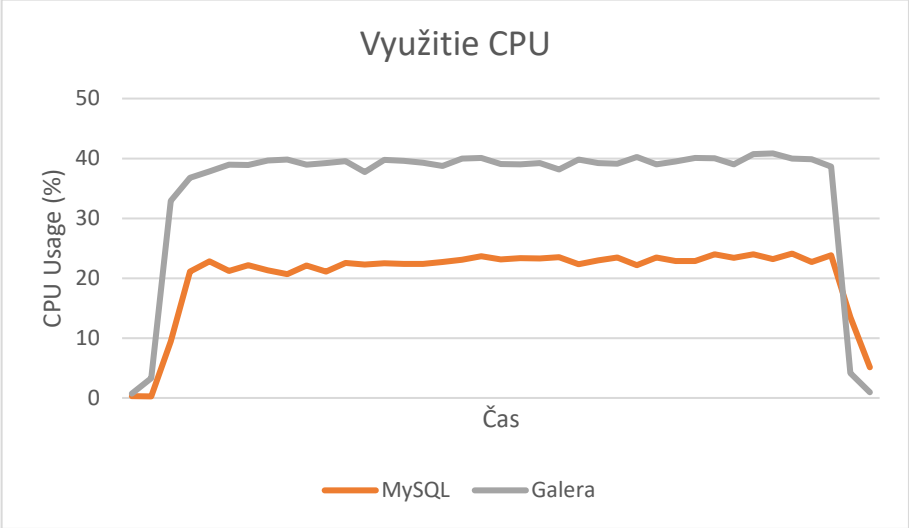
Zdroj: vlastné spracovanie

*Graf 4: Zaťaženie CPU pri porovnaní MySQL a MySQL Galera Cluster*



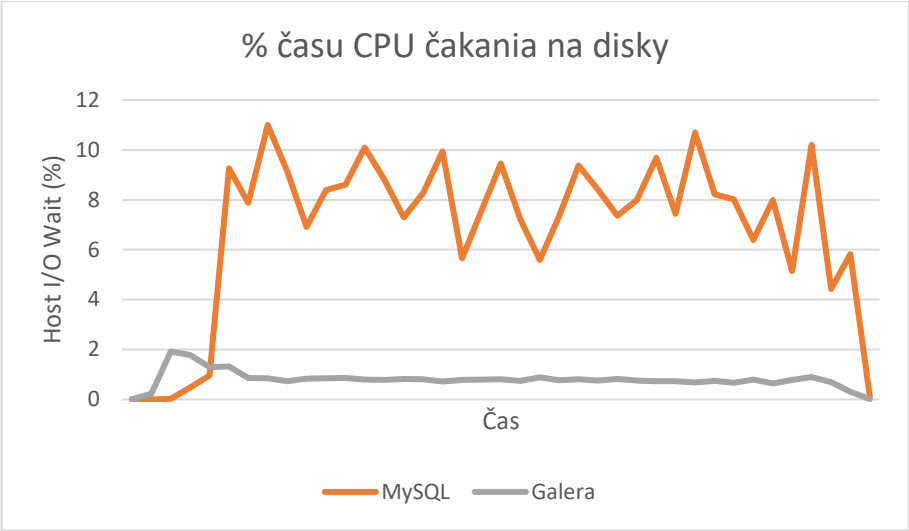

Zdroj: vlastné spracovanie

*Graf 5: Využitie CPU pri porovnaní MySQL a MySQL Galera Cluster*



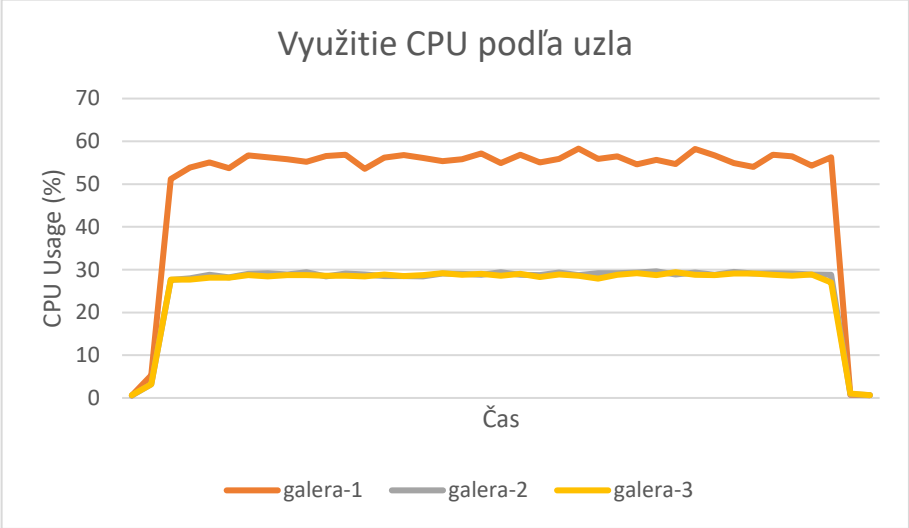

Zdroj: vlastné spracovanie

*Graf 6: Percento čakania na disky pri porovnaní MySQL a MySQL Galera Cluster*




Zdroj: vlastné spracovanie

*Graf 7: Využitie CPU MySQL Galera Clusterom rozdelené podľa uzlov*




Zdroj: vlastné spracovanie

Z výsledkov implementácie *MySQL Galera Clusteru* sme mohli pozorovať, že vytvorenie distribuovaného systému nemalo žiadny zásadný vplyv na rýchlosť v porovnaní s nedistribuovanou verziou *MySQL*. Nakoľko ide o pomerne nízke percento, považujeme tento vplyv za štatisticky nevýznamný.

Na grafoch sme mohli pozorovať ďalšie dôležité informácie. Jednou z nich je zvýšenie využitia CPU na takmer dvojnásobné hodnoty. Navýšenie využitia CPU je očakávané, nakoľko sa všetky repliky nachádzajú na jednom fyzickom serveri. Taktiež sme mohli

pozorovať zníženie čakania odozvy diskov, ktorá poklesla o niekoľko percent a v prípade *Galera Clusteru* dosahovala stabilné hodnoty, zatiaľ čo v prípade *MySQL* dosahovala kolísavé hodnoty. Predpokladali sme, že toto zníženie odozvy mohlo byť spôsobené súborovom systémom *ZFS*, ktorý mal možnosť udržať viacero dát z databázy v pamäti RAM a tým pádom nebolo nutné vykonávať čítanie dát z diskového poľa.

Ďalší priestor na optimalizáciu distribuovaného databázového systému sme mohli vidieť v poslednom grafe, ktorý nám ukazoval využitie zdrojov CPU podľa jednotlivých uzlov v databázovom systéme. Nakoľko aktuálne nemáme implementované rozdeľovanie záťaže naprieč uzlami, všetky dopyty vykonávané počas testov smerovali na databázový uzol *galera-1*, ktorý bol vzhľadom na to aj najvyťaženejší.

### *4.1.3 Distribúcia záťaže*

Ako ďalší krok v rámci našej optimalizácie sme sa rozhodli implementovať *ProxySQL* do *Galera Clusteru*. Pri tomto kroku sme do nášho testovacieho systému pridali ďalší virtuálny stoj na ktorom sa nachádzal iba softvér *ProxySQL*. Ten sme nainštalovali podľa oficiálneho postupu za pomoci nasledujúcich príkazov:

```
apt-get install -y --no-install-recommends lsb-release wget apt-transport-https ca-
certificates gnupg
wget -O - 'https://repo.proxysql.com/ProxySQL/proxysql-2.7.x/repo_pub_key' | apt-key add -
echo deb https://repo.proxysql.com/ProxySQL/proxysql-2.7.x/$(lsb_release -sc)/ ./ | tee
/etc/apt/sources.list.d/proxysql.list
apt-get update
apt-get install proxysql mysql-client
```

Po inštalácii sme sa ku ProxySQL serveru pripojili za použitia príkazu:

```
mysql -u admin -padmin -h 127.0.0.1 -P6032 --prompt='Admin> '
```

A za pomoci SQL dopytov nakonfigurovali *ProxySQL* server tak, aby dopyty posielal na existujúci *Galera Cluster*. Pre správne fungovanie bolo potrebné pridať servery na ktoré sa má pripájať do tabuľky *mysql_servers* a užívateľa do tabuľky *mysql_users*. Následne po uložení zmien začal *ProxySQL* preposielať dopyty na uvedené servery.

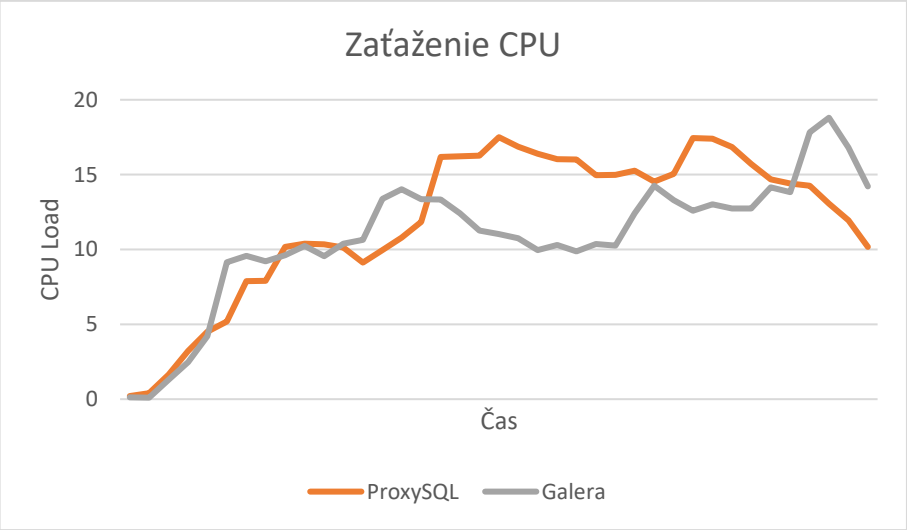
```
INSERT INTO mysql_servers(hostgroup_id,hostname,port) VALUES (1,'10.10.0.58',3306);
INSERT INTO mysql_servers(hostgroup_id,hostname,port) VALUES (1,'10.10.0.74',3306);
INSERT INTO mysql_servers(hostgroup_id,hostname,port) VALUES (1,'10.10.0.24',3306);
INSERT INTO mysql_users(username,password,default_hostgroup) VALUES ('user','password',1);
LOAD MYSQL SERVERS TO RUNTIME;
LOAD MYSQL USERS TO RUNTIME;
SAVE MYSQL USERS TO DISK;
SAVE MYSQL SERVERS TO DISK;
```

*Tabuľka 3: Porovnanie spracovaných dopytov medzi Galera Clusterom bez a s ProxySQL*

| *Druh testu* | *ProxySQL* | *Galera* |
|---|---|---|
| *Delivery* | 1682 | 1642 |
| *New Order* | 21323 | 18983 |
| *Order Status* | 2002 | 1628 |
| *Payment* | 18310 | 17947 |
| *Stock Level* | 1981 | 1712 |
| *Vážený priemer* | *3539,05* | *3291,768* |

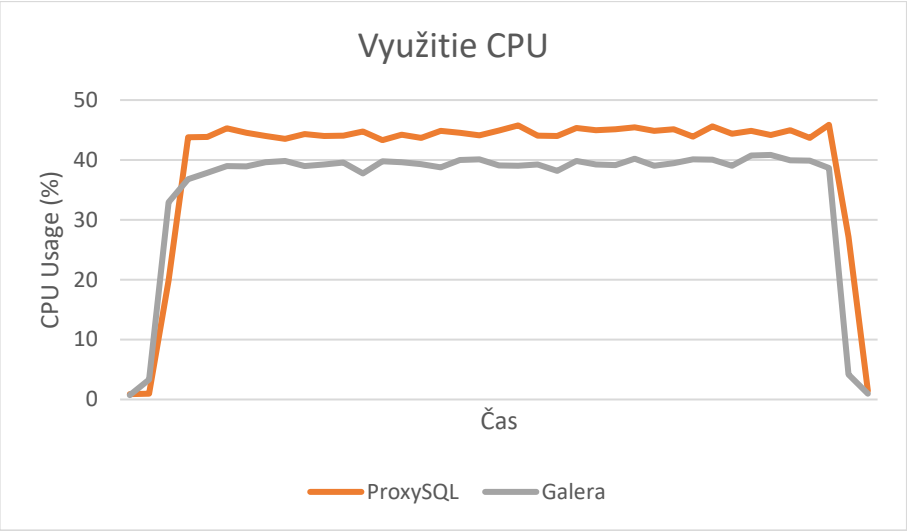
Zdroj: vlastné spracovanie

*Graf 8: Zaťaženie CPU pri porovnaní Galera Clusteru bez a s ProxySQL*



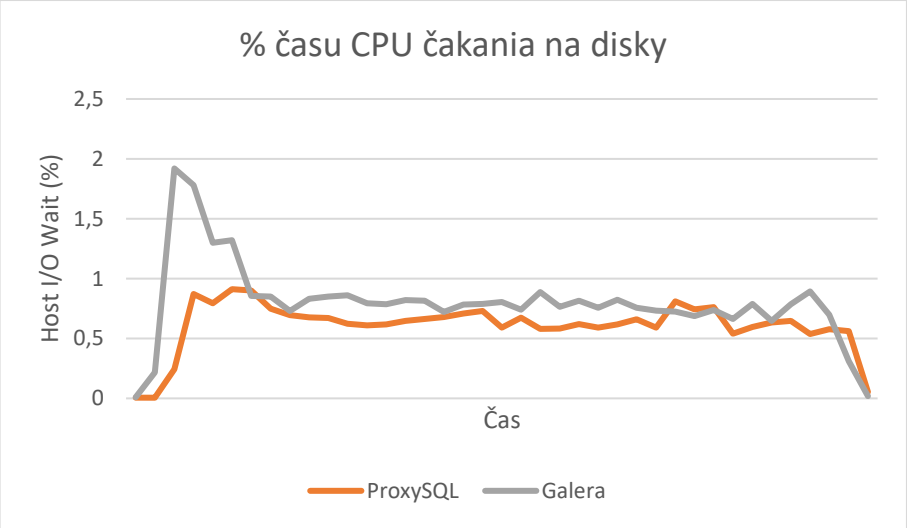

Zdroj: vlastné spracovanie

*Graf 9: Využitie CPU pri porovnaní Galera Clusteru bez a s ProxySQL*



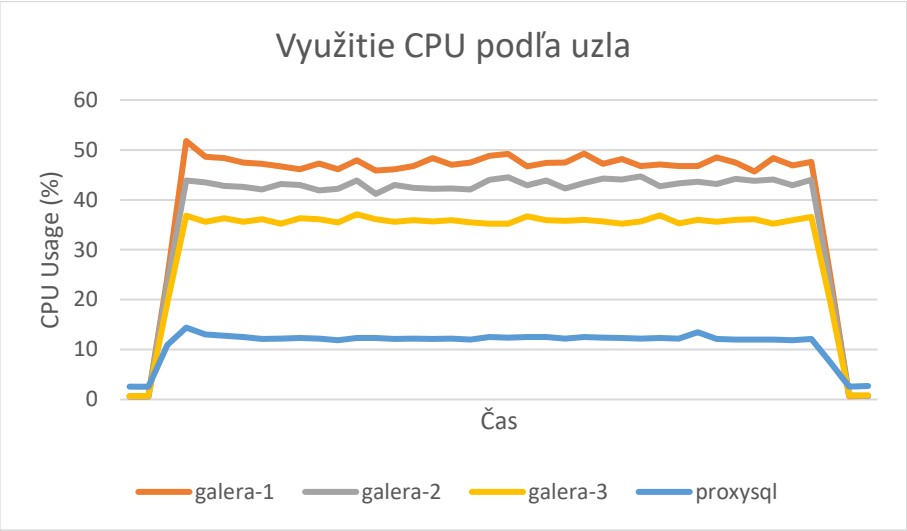

Zdroj: vlastné spracovanie

*Graf 10: Percento času čakania na disky pri porovnaní Galera Clusteru bez a s ProxySQL*




Zdroj: vlastné spracovanie

*Graf 11: Využitie CPU podľa uzla pri implementácii ProxySQL do Galera Clusteru*




Zdroj: vlastné spracovanie

Po implementácii *ProxySQL* sme mohli pozorovať zvýšenie počtu spracovaných dopytov o cca 7%. Nakoľko *ProxySQL* nám umožňoval rovnomerne rozdeliť dopyty naprieč viacerými uzlami, spracovanie dopytov mohlo prebiehať paralelnejšie na viacerých uzloch, hlavne v prípade dopytov, ktoré nezapisovali do databázy nové dáta alebo nad nimi nevykonávali úpravy. Paralelnosť sme mohli pozorovať aj v grafe využitia CPU podľa uzla, kedy v porovnaní s grafom *Galera Clusteru* bez *ProxySQL* bolo využitie CPU jednotlivých uzlov vyrovnanejšie. Ostatné štatistiky zaťaženia systému sa nám výrazne nezmenili.

### *4.1.4 Vplyv sieťového spojenia na systém*

Nakoľko jednotlivé uzly spolu komunikujú po sieti, pozreli sme sa aj na dopad rôznych sieťových komplikácií na chod distribuovaného databázového systému. Na tento účel sme využili sadu nástrojov *NetEm* a *tc*, ktoré sme si predstavili v predchádzajúcich častiach tejto práce.

Zamerali sme sa na testovanie dvoch sieťových komplikácii s ktorými sa môžeme najčastejšie stretnúť a to strata paketov v sieti a vysoká odozva na spojení. Obe tieto komplikácie sme simulovali na odchádzajúcom spojení z *ProxySQL* servera smerom na *Galera Cluster*. Komplikácie sme simulovali za použitia príkazov:

```
1.  tc qdisc add dev eth0 root netem loss 5%
2.  tc qdisc add dev eth0 root netem loss 10%
3.  tc qdisc add dev eth0 root netem loss 25%
```

Tento príkaz pridal na odchádzajúce spojenia sieťovým adaptérom *eth0* stratu paketov vo výške 5% a 7% všetkých odchádzajúcich paketov.

```
1.  tc qdisc add dev eth0 root netem delay 25ms
2.  tc qdisc add dev eth0 root netem delay 50ms
3.  tc qdisc add dev eth0 root netem delay 100ms
4.  tc qdisc add dev eth0 root netem delay 250ms
```

Vyššie uvedený príkaz pridal na odchádzajúce spojenia sieťovým adaptérom *eth0* odozvu vo výške 25, 50, 100 a 250 milisekúnd.

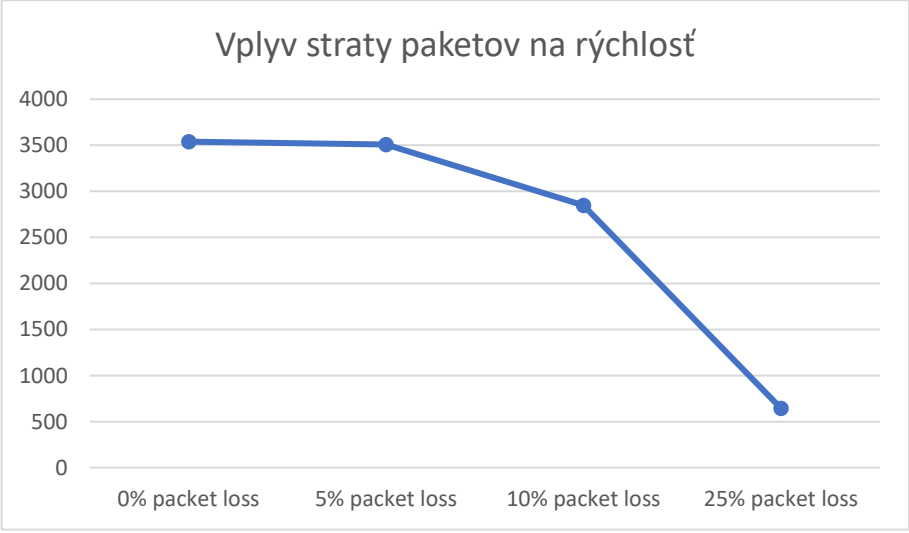
Po aplikácii jednotlivých príkazov sme vykonali testovanie a dostali nasledovné výsledky:

*Tabuľka 4: Vplyv straty paketov na chod databázového systému*

| | | *Strata paketov* | | |
|---|---|---|---|---|
| *Druh testu* | Základné hodnoty | 5% | 10% | 25% |
| *Delivery* | 1682 | 1745 | 1419 | 348 |
| *New Order* | 21323 | 20791 | 16752 | 3665 |
| *Order Status* | 2002 | 1848 | 1474 | 353 |
| *Payment* | 18310 | 18537 | 15173 | 3573 |
| *Stock Level* | 1981 | 1874 | 1471 | 358 |
| *Vážený priemer* | *3539,05* | *3509,11* | *2847* | *646* |

Zdroj: vlastné spracovanie

*Graf 12: Vplyv straty paketov na rýchlosť spracovania dopytov*




Zdroj: vlastné spracovanie

Ako môžeme vidieť, strata paketov vo výške 5% nemala významnejší dopad na funkčnosť distribuovaného systému. Podobne ako pri nasadzovaní *Galera Clusteru* tak aj v tomto prípade sme považovali túto hodnotu za štatisticky nevýznamnú odchýlku, ktorá vznikla pri testovaní. Avšak akonáhle strata paketov dosiahla úroveň 10%, mohli sme pozorovať pokles vo výpočtovej rýchlosti databázového systému. V momente kedy strata paketov dosiahla 25% výpočtový výkon databázového systému prakticky klesol na minimum a v tomto stave by databázový systém bolo možné využívať iba s ťažkosťami.

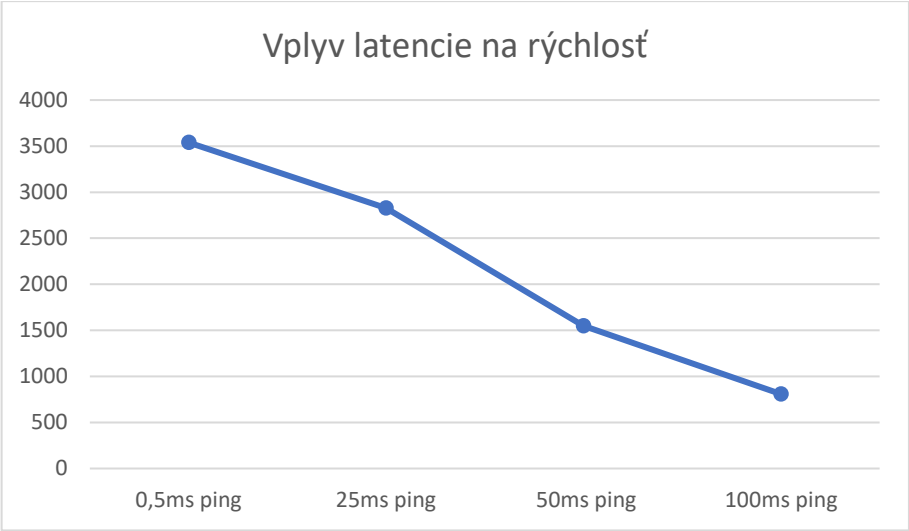
Vzhľadom na výsledok testovania sme vyvodili záver, že ak je strata paketov do výšky 5%, neovplyvnila chod databázového systému do zásadnej miery a teda bola pre nás prijateľná aj keď nie žiadaná.

*Tabuľka 5: Vplyv latencie na chod databázového systému*

| | | *Latencia* | | | |
|---|---|---|---|---|---|
| *Druh testu* | Základné hodnoty | 25ms | 50ms | 100ms | 250ms |
| *Delivery* | 1682 | 1441 | 740 | 398 | 159 |
| *New Order* | 21323 | 16198 | 9213 | 4832 | 2040 |
| *Order Status* | 2002 | 1449 | 819 | 428 | 187 |
| *Payment* | 18310 | 15507 | 8135 | 4225 | 1498 |
| *Stock Level* | 1981 | 1510 | 824 | 477 | 203 |
| *Vážený priemer* | *3539,05* | *2827* | *1547,844* | *808,654* | *316,82* |

Zdroj: vlastné spracovanie

*Graf 13: Vplyv latencie na rýchlosť spracovania dopytov*

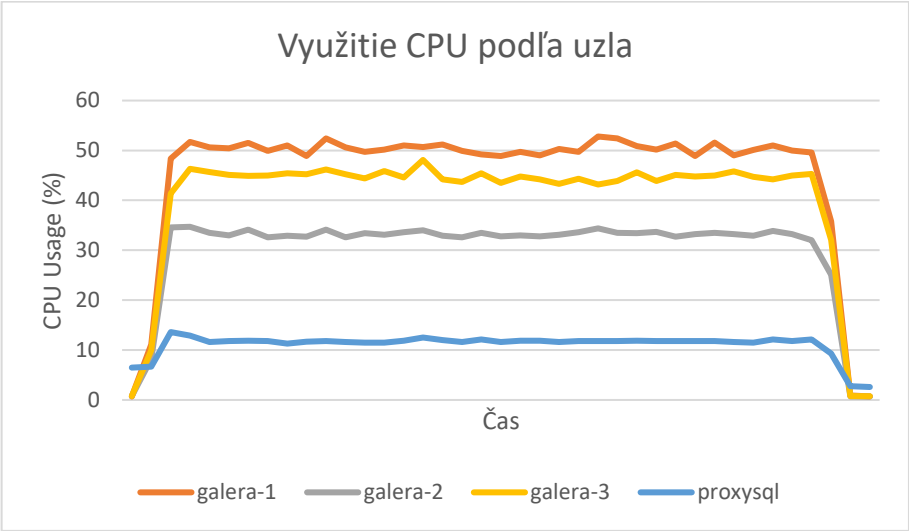



Zdroj: vlastné spracovanie

Testy so základnými hodnotami boli vykonávané pri nameranej latencii 0,5ms. Z výsledkov testov sme mohli skonštatovať, že latencia má zásadný dopad na chod distribuovaného databázového servera. Vzhľadom na výsledky vplyvu, bolo nutné udržať latenciu na čo najnižšej úrovni aby nebol databázový systém negatívne ovplyvňovaný.

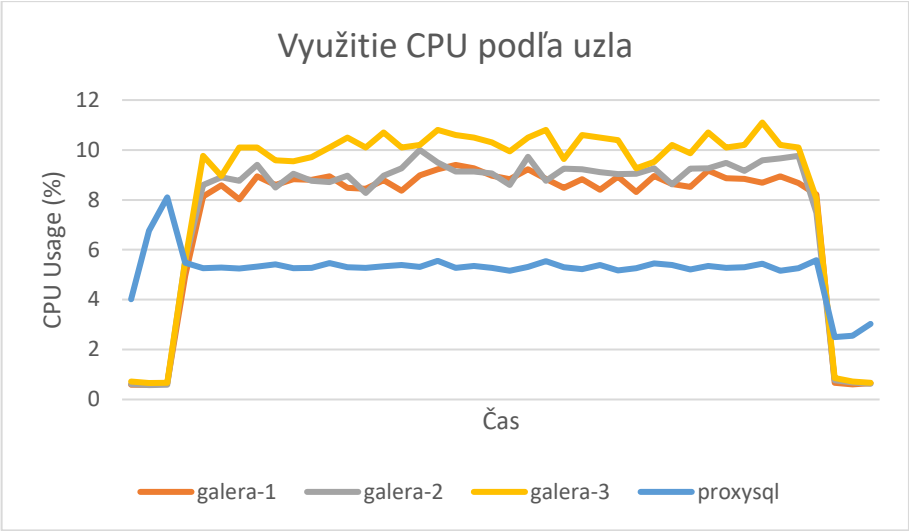
Pozreli sme sa aj na grafy využitia CPU podľa uzla pri strate paketov vo výške 5% a odozve 250ms.

*Graf 14: Využitie CPU podľa uzla pri 5% strate paketov*




Zdroj: vlastné spracovanie

*Graf 15: Využitie CPU podľa uzla pri 250ms odozve*




Zdroj: vlastné spracovanie

Z pozorovania využitia CPU na jednotlivých uzloch pri rôznych sieťových situáciách sme mohli pozorovať pokles využitia CPU až o 20 až 40 % na rôznych uzloch pri vysokej sieťovej odozve. Pri porovnaní využitia zdrojov pri strate paketov 5% a bez straty paketov nepozorujeme žiadne výrazne zmeny vo využití zdrojov databázového systému. Nakoľko využitie CPU bolo do veľkej miery ovplyvnené počtom vykonaných dopytov a stavom sieťového spojenia, pre ostatné situácie sme neuvádzali tabuľky využitia CPU.

### *4.1.5 Vplyv indexácie tabuliek na rýchlosť*

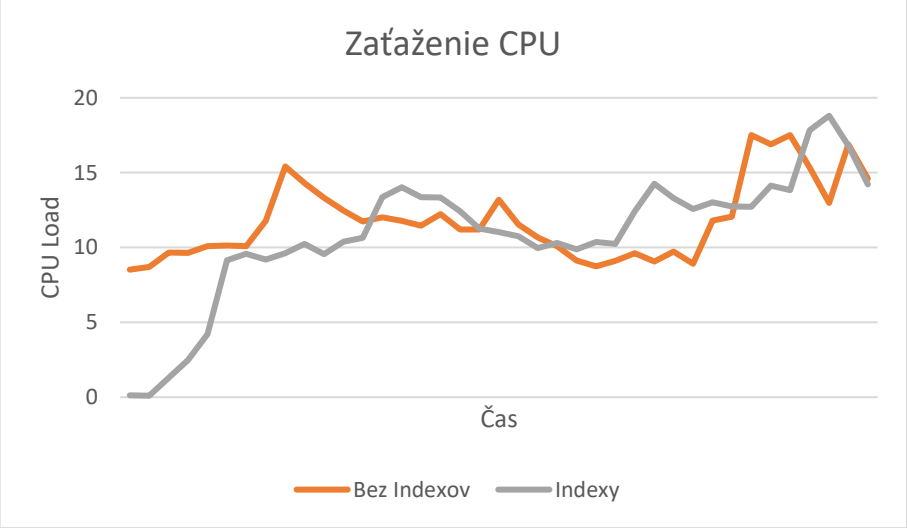
Než sme prešli ku samotnej softvérovej optimalizácii databázového systému, rozhodli sme sa pozrieť na vplyv indexov v našom systéme. Ako sme spomínali v predchádzajúcich častiach práce, indexovanie záznamov môže mať v niektorých prípadoch negatívny vplyv na chod databázového systému. Cieľom tohto testu bolo vyhodnotiť či je vhodné využiť indexáciu v databáze ak sme brali do úvahy cieľ tejto práce a to návrh a optimalizácia databázy pre stredný až veľký podnik.

*Tabuľka 6: Vplyv indexácie na rýchlosť spracovania dopytov*

| *Druh testu* | *Indexácia* | *Bez indexácie* |
|---|---|---|
| *Delivery* | 1682 | 2283 |
| *New Order* | 21323 | 25796 |
| *Order Status* | 2002 | 2299 |
| *Payment* | 18310 | 24989 |
| *Stock Level* | 1981 | 2341 |
| *Vážený priemer* | *3539,05* | *4387,812* |

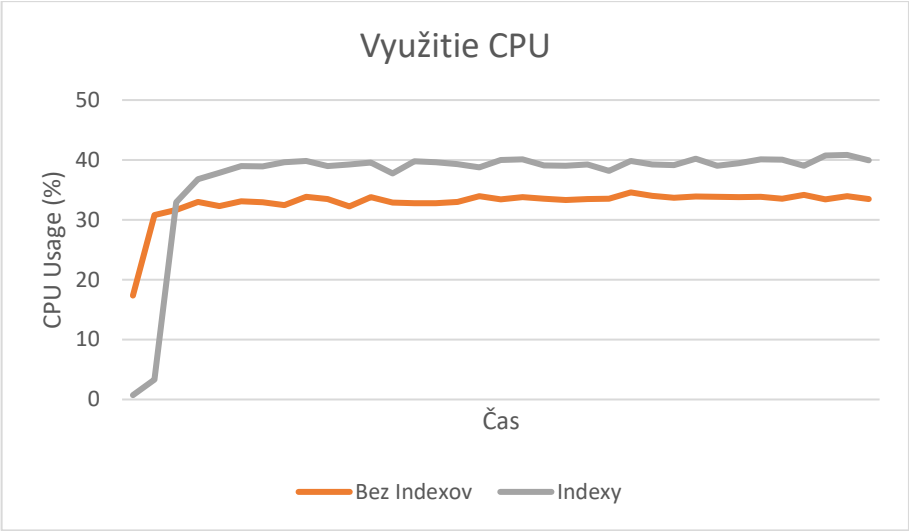
Zdroj: vlastné spracovanie

*Graf 16: Zaťaženie CPU pri dopadoch indexácie*



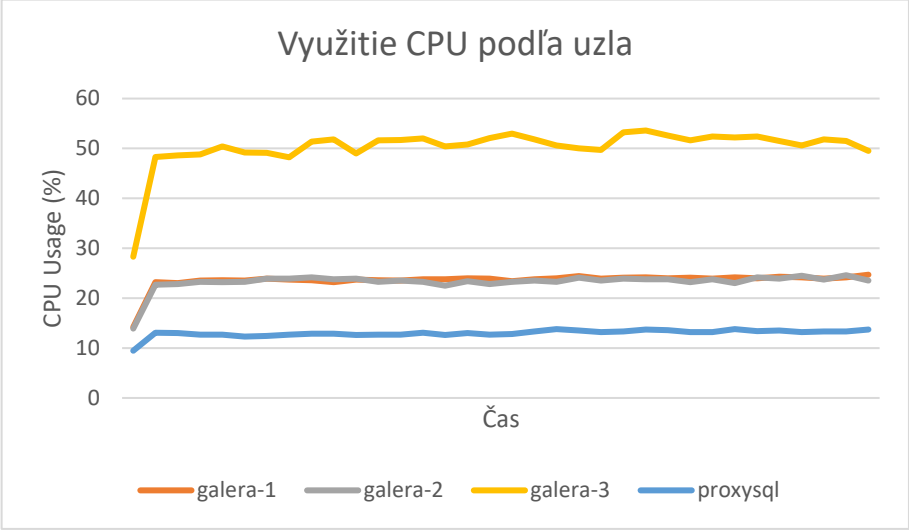

Zdroj: vlastné spracovanie

*Graf 17: Využitie CPU pri dopadoch indexácie*




Zdroj: vlastné spracovanie

*Graf 18: Využitie CPU podľa uzla pri odstránení indexácie*




Zdroj: vlastné spracovanie

Z výsledkov testovania databázového systému bez indexácie môžeme vidieť značný nárast spracovaných dopytov až o 21% zatiaľ čo zaťaženie CPU ostalo bez výraznejšej zmeny a celkové využitie CPU kleslo z približne 40% na 35%. Nakoľko testovací nástroj DBT2 vytváral tabuľky s indexami a zároveň indexácia tabuliek je veľmi špecifiká vzhľadom na ich obsah a využitie, po dobu ostatných testov sme vykonávali všetky testovania s indexáciou tabuliek, ak nebolo spomenuté inak.

### *4.1.6 Zhrnutie prvej časti optimalizácie*

Naprieč testovaním sme zistili niekoľko skutočností, ktoré sme následne aplikovali v ďalších fázach optimalizácie databázového systému. Ako prvé sme mohli vidieť, že nasadenie distribuovaného databázového systému nemalo takmer žiadny vplyv na prevádzku a výsledky testovania. Ako druhé sme sa mohli dozvedieť, že pridanie softvéru na distribúciu záťaže pomohlo zlepšiť výkon databázového systému, nakoľko sa záťaž nesústreďovala iba na jeden uzol ale bola rozložená naprieč všetky uzly systému. Následne sme sa dozvedeli, že sieťová odozva je dôležitým faktorom, ktorý zásadne ovplyvňuje chod distribuovaného databázového systému, zatiaľ čo strata paketov nepredstavuje až tak vysoké ohrozenie chodu systému. Ako posledné sme zistili, že odstránením indexácie z tabuliek dokážeme zvýšiť výkon systému, vzhľadom na veľké množstvo zápisov, ktoré sú nad tabuľkami vykonávané. Nižšie uvádzame tabuľku v ktorej sú zhrnuté najpodstatnejšie zmeny a ich vplyv na distribuovaný databázový systém:

*Tabuľka 7: Výsledky prvej optimalizácie*

| *Druh testu* | *MySQL* | *Galera* | *Galera + ProxySQL* | *Odstránenie indexácie* |
|---|---|---|---|---|
| *Delivery* | 1682 | 1642 | 1682 | 2283 |
| *New Order* | 18604 | 18983 | 21323 | 25796 |
| *Order Status* | 1601 | 1628 | 2002 | 2299 |
| *Payment* | 17847 | 17947 | 18310 | 24989 |
| *Stock Level* | 1668 | 1712 | 1981 | 2341 |
| *Výsledok* | *3248,8* | *3291,768* | *3539,05* | *4387,812* |

Zdroj: vlastné spracovanie

Vo výsledku prvých krokov optimalizácie databázového systému môžeme sledovať nárast v počte spracovaných dopytov o 28% (porovnanie medzi stĺpcami „*Galera*“ a „*Odstránenie indexácie*“). Stĺpec „*MySQL*“ predstavuje základný *MySQL* databázový server bez, ktorý nie je distribuovaný. Najväčšiu zásluhu na zvýšení výkonu malo odstránenie indexácie, avšak aj implementácia distribúcie záťaže priniesla benefit vo výške 7%.

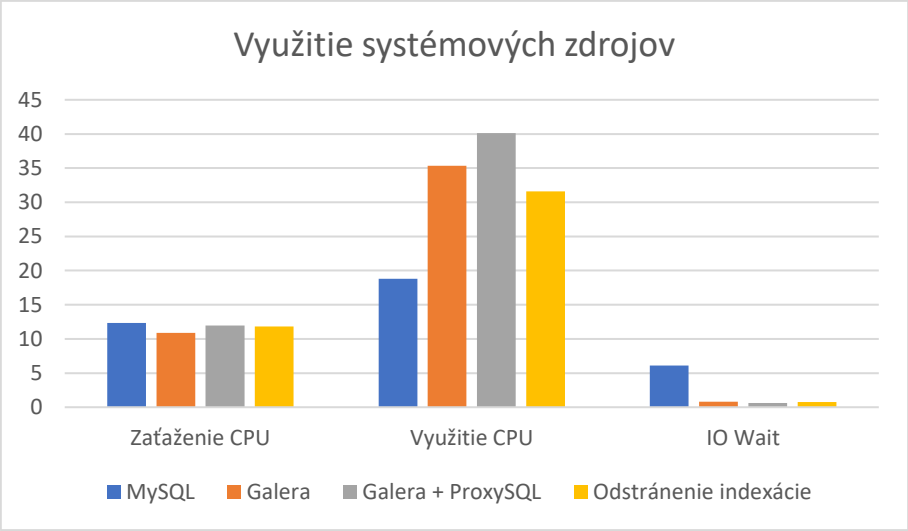
Taktiež sme mohli sledovať aj vplyv najpodstatnejších zmien na využitie systémových zdrojov:

Tabuľka 8: Využitie systémových zdrojov po prvej optimalizácii

| | *MySQL* | *Galera* | *Galera + ProxySQL* | *Odstránenie indexácie* |
|---|---|---|---|---|
| *Zaťaženie CPU* | 12,316744 | 10,86 | 11,94846154 | 11,82282051 |
| *Využitie CPU* | 18,805909% | 35,348% | 40,10589744% | 31,59972973% |
| *IO Wait* | 6,0827516% | 0,797% | 0,609455897% | 0,743354054% |

Zdroj: vlastné spracovanie

Graf 19: Využitie systémových zdrojov po prvej optimalizácii




Zdroj: vlastné spracovanie

Z týchto údajov sme mohli skonštatovať, že naše zmeny mali minimálny vplyv na priemerné zaťaženie CPU, avšak pridávanie dodatočných softvérových súčastí zvyšovalo priemerné využitie CPU až do momentu, kedy sme neodstránili z tabuľky indexáciu. Vtedy prišlo ku opätovnému poklesu využitia CPU. Taktiež sme mohli pozorovať, že okrem nedistribuovanej databázy náš server nepozoroval žiadne spomalenie výkonu z dôvodu čakania na diskové polia. V závere sme mohli povedať, že náš server má ešte dostatok dostupných zdrojov na dodatočnú optimalizáciu databázového systému.

## 4.2 Testovanie zmien konfigurácie MySQL Galera Clusteru

V tejto fázy optimalizácie distribuovaného databázového systému *MySQL Galera Cluster* sme sa pozreli na dokumentáciu *MySQL* a *MySQL Galera Cluster*, podľa ktorej sme nastavovali rôzne premenné databázového systému za cieľom hlbšej optimalizácie databázového systému. Za počiatočné výsledky uvádzané v tabuľkách tejto kapitoly považujeme výsledky ktoré sme dosiahli vytvoreným distribuovanej databázy a nasadením

*ProxySQL* systému na distribúciu záťaže. V tejto časti práce sme vo väčšine prípadov neuvádzali vplyv na zdroje CPU, nakoľko rozdiel vo využití zdrojov bol iba minimálny. V prípade, že prišlo ku zásadnejšiemu rozdielu vo využití systémových zdrojov, bol uvedený aj graf využitia CPU pre jednotlivé uzly. Premenné ktoré sme v rámci optimalizácie systému menili, boli podrobnejšie opísané v prvej časti tejto práce.

### *4.2.1 Vykonávanie zmien na základe výstupov*

V prvom kroku sme sa pozreli na výstup v systémovom súbore umiestnenom v lokalite */var/log/mysql/error.log*. V tomto súbore sme našli upozornenie, ktoré odporúčalo navýšiť hodnotu premennej *innodb_redo_log_capacity*.

```
[Warning] [MY-013865] [InnoDB] Redo log writer is waiting for a new redo log file. Consider
increasing innodb_redo_log_capacity.
```

Ako bolo spomenuté v predchádzajúcich častiach práce, tzv. *redo log* slúži na obnovu dát pri neúspešných SQL transakciách. Pre otestovanie vplyvu na rýchlosť databázového systému sme sa rozhodli nastaviť veľkosť súboru na 4Gb, 8Gb a 16Gb pridaním nasledovných riadkov do konfigurácie MySQL, kde hodnota *X* odpovedala maximálnej veľkosti súborov v bajtoch:

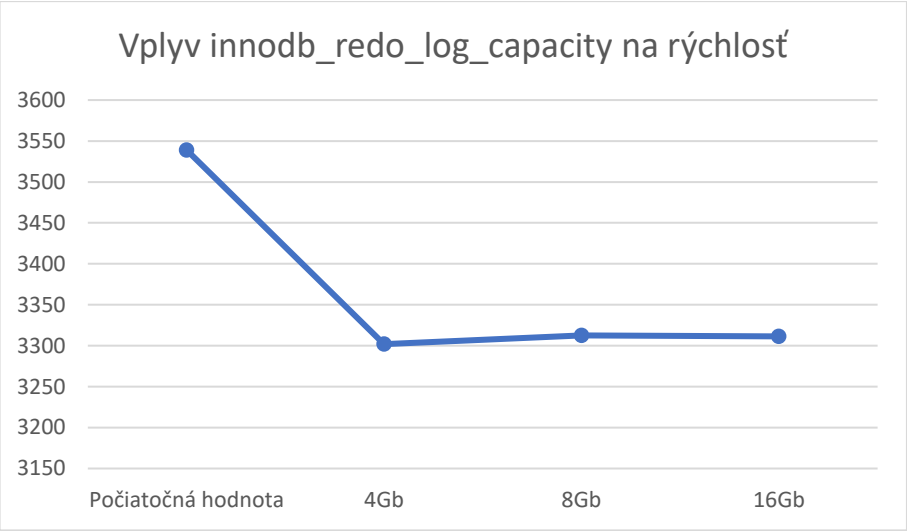
```
[server]
innodb_redo_log_capacity = X
```

*Tabuľka 9: Vplyv innodb_redo_log_capacity na rýchlosť*

| | | *innodb_redo_log_capacity* | | |
|---|---|---|---|---|
| *Druh testu* | Počiatočné hodnoty | 4Gb | 8Gb | 16Gb |
| *Delivery* | 1682 | 1682 | 1714 | 1750 |
| *New Order* | 21323 | 19167 | 19660 | 19343 |
| *Order Status* | 2002 | 1802 | 1732 | 1759 |
| *Payment* | 18310 | 17849 | 17461 | 17769 |
| *Stock Level* | 1981 | 1746 | 1734 | 1773 |
| *Výsledok* | *3539,05* | *3301,884* | *3312,486* | *3311,26* |

Zdroj: vlastné spracovanie

*Graf 20: Vplyv innodb_redo_log_capacity na rýchlosť*




Zdroj: vlastné spracovanie

Avšak napriek odporúčaniam z hlášky MySQL servera sme mohli pozorovať, že úprava tejto premennej mala iba negatívny vplyv v rozmedzí 6 až 7%. Vzhľadom na to sme sa rozhodli neaplikovať zmeny v konfigurácii tejto premennej v ďalších optimalizáciách databázových serverov.

### *4.2.2 Optimalizácia pamäte RAM*

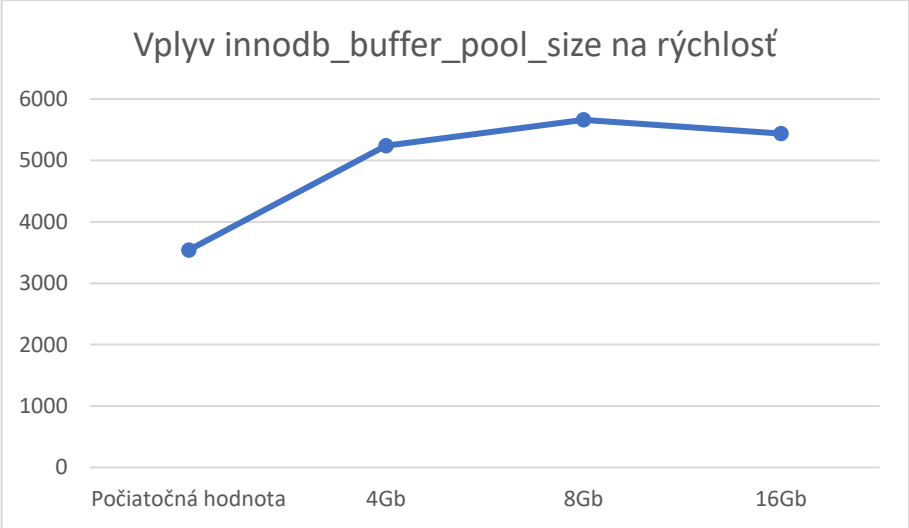
Nasledovne sme sa pozreli na možnosť lepšieho využitia pamäte RAM na databázových systémoch. Na správu pamäte ponúka databázový úložný systém *InnoDB* premennú *innodb_buffer_pool_size*. Pre pripomenutie, jednotlivé databázové servery majú k dispozícii 64GB pamäte RAM. Testy sme vykonávali s hodnotami 16GB, 32GB a 48GB dostupnej pamäte RAM pre *InnoDB*. Test so 64GB RAM sme nevykonávali, nakoľko sme časť pamäte RAM ponechávali pre operačný systém a chod *MySQL* servera.

*Tabuľka 10: Vplyv innodb_buffer_pool_size na rýchlosť*

| | | *innodb_buffer_pool_size* | | |
|---|---|---|---|---|
| *Druh testu* | Počiatočné hodnoty | 16Gb | 32Gb | 48Gb |
| *Delivery* | 1682 | 2594 | 3011 | 2761 |
| *New Order* | 21323 | 30183 | 33907 | 31642 |
| *Order Status* | 2002 | 2650 | 3062 | 2895 |
| *Payment* | 18310 | 28612 | 29473 | 29310 |
| *Stock Level* | 1981 | 2697 | 3161 | 2747 |
| *Výsledok* | *3539,05* | *5240,63* | *5660,18* | *5435,664* |

Zdroj: vlastné spracovanie

*Graf 21: Vplyv innodb_buffer_pool_size na rýchlosť*




Zdroj: vlastné spracovanie

Po testoch môžeme vidieť nárast výkonu systému až o 60% pri využití 32GB pamäte RAM. Pridelenie viac ako 32GB RAM nemalo už žiadny zásadnejší význam na chod databázového systému (mohli sme pozorovať mierny pokles so výkone) a vzhľadom na to sme sa v ďalších krokoch rozhodli pracovať s hodnotou 32GB.

### *4.2.3 Optimalizácia kapacity IO úloh*

Taktiež sme sa pokúsili optimalizovať databázu vzhľadom na rýchlosť diskového poľa, ktoré sme mali na serveri. Pri úprave tohto nastavenia sme brali do úvahy, že disky sa nachádzali v ZFS softvérovom RAID, teda všetky dáta museli byť zapísané súbežne na 2

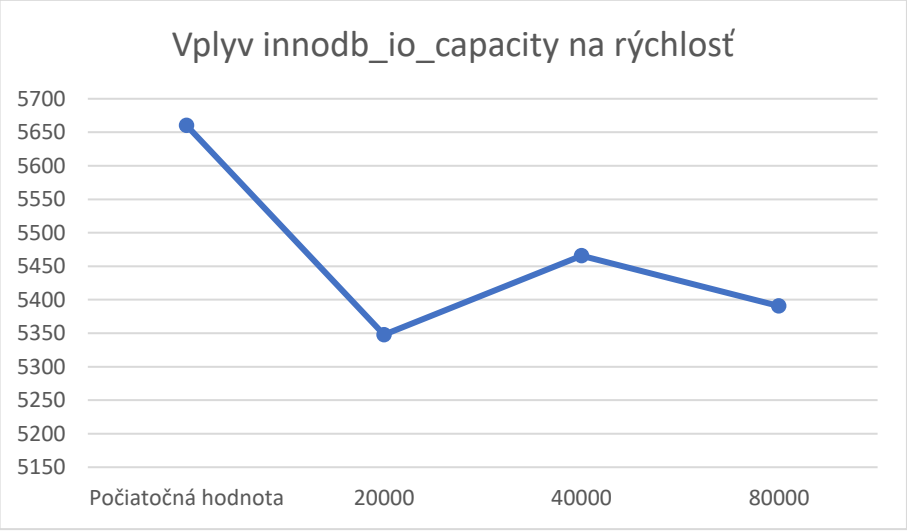
disky, čo mohlo negatívne ovplyvňovať rýchlosť diskového poľa výmenou za väčšiu kapacitu a redundanciu.

*Tabuľka 11: Vplyv innodb_io_capacity na rýchlosť*

| | | *innodb_io_capacity* | | |
|---|---|---|---|---|
| *Druh testu* | Počiatočné hodnoty | 20000 | 40000 | 80000 |
| *Delivery* | 3011 | 2670 | 2861 | 2760 |
| *New Order* | 33907 | 30565 | 31728 | 31638 |
| *Order Status* | 3062 | 2628 | 2727 | 2756 |
| *Payment* | 29473 | 29452 | 29576 | 28793 |
| *Stock Level* | 3161 | 2712 | 2730 | 2869 |
| *Výsledok* | *5660,18* | *5347,802* | *5465,6* | *5390,698* |

Zdroj: vlastné spracovanie

*Graf 22: Vplyv innodb_io_capacity na rýchlosť*




Zdroj: vlastné spracovanie

Po vykonaní testov sa náš predpoklad potvrdil a navýšenie maximálneho počtu IO operácii za sekundu malo skôr negatívny vplyv na chod databázového systému. Mohli sme pozorovať pokles výpočtového výkonu o cca 5-6%, ktorý sme mohli považovať za štatistickú odchýlku v rámci testovania, avšak nakoľko všetky testy pri všetkých testovaných hodnotách dosahovali negatívne výsledky, rozhodli sme sa toto nastavenie nezakomponovať do ďalšej fázy optimalizácie.

### *4.2.4 Optimalizácia InnoDB vláken*

Virtuálne stroje na ktorých sa nachádzali jednotlivé uzly databázového systému mali k dispozícii 8 jadier. Vzhľadom na to sme sa rozhodli testovať aj vplyv manuálneho nastavenia vláken na chod systému.

Ako sme spomínali v predchádzajúcich častiach, predvolená hodnota tejto premennej je závislá na počte CPU jadier systému a vypočíta sa ako:

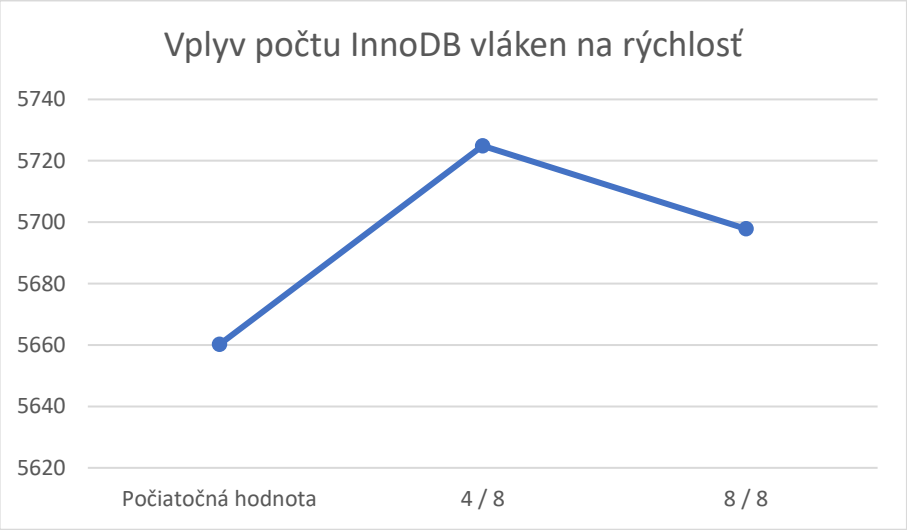
$$hodnota = \frac{počet\ logických\ procesorov}{2} \qquad hodnota \geq 4$$

*Tabuľka 12: Vplyv počtu vláken na rýchlosť*

| | | *innodb_read/write_io_threads* | |
|---|---|---|---|
| *Druh testu* | Počiatočné hodnoty | 8/4 | 8/8 |
| *Delivery* | 3011 | 3006 | 3023 |
| *New Order* | 33907 | 34169 | 34259 |
| *Order Status* | 3062 | 3050 | 3128 |
| *Payment* | 29473 | 29969 | 29542 |
| *Stock Level* | 3161 | 2987 | 3091 |
| *Výsledok* | *5660,18* | *5724,888* | *5697,858* |

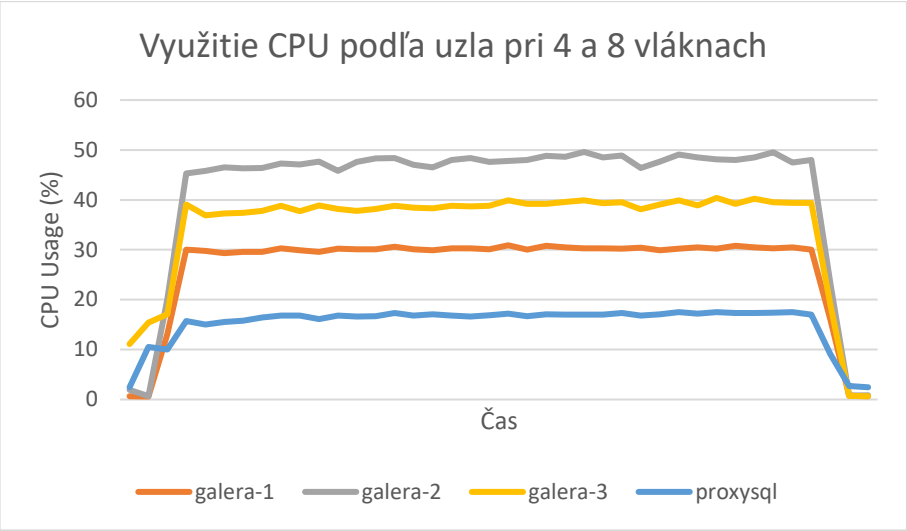
Zdroj: vlastné spracovanie

*Graf 23: Vplyv počtu InnoDB vláken na rýchlosť*



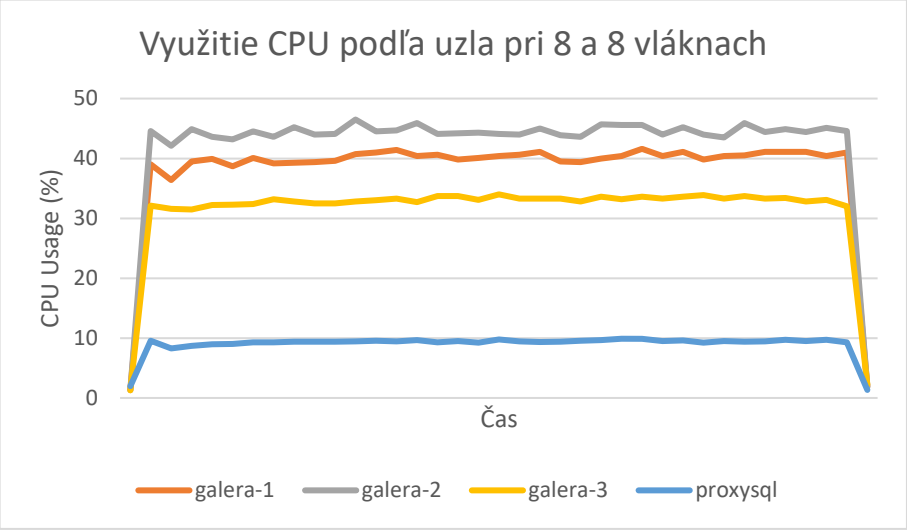

Zdroj: vlastné spracovanie

*Graf 24: Využitie CPU podľa uzla pri 4 vláknach zápisu a 8 vláknach čítania*




Zdroj: vlastné spracovanie

*Graf 25: Využitie CPU podľa uzla pri 8 vláknach zápisu a 8 vláknach čítania*




Zdroj: vlastné spracovanie

Vo výsledku sme mohli pozorovať zanedbateľné zisky vo výkone pri navýšení počtu vláken zápisu zo 4 na 8 a pokles týchto ziskov pri navýšení vláken čítania. Aj napriek tomu, že išlo o zanedbateľný rozdiel rozhodli sme sa úpravu nastavenia ponechať do ďalších krokov optimalizácie.

### *4.2.5 Optimalizácia wsrep vláken*

Optimalizácia *wsrep* vláken bola jednou z mála optimalizácii, ktorú bolo možné vykonať na úrovni distribučného systému a nie nad databázovým systémom. Predvolene

bola replikácia zabezpečovaná iba jedným vláknom, čo obmedzovalo jej výkon na systémoch s viacerými logickými jadrami procesorov. Keďže náš systém mál 8 jadier na každom zo serverov, rozhodli sme sa jej hodnotu nastaviť na 8.

*Tabuľka 13: Vplyv 8 wsrep vláken na rýchlosť*

| *Druh testu* | *Počiatočné hodnoty* | *8 wsrep vláken* |
|---|---|---|
| *Delivery* | 3006 | 6669 |
| *New Order* | 34169 | 75357 |
| *Order Status* | 3050 | 6747 |
| *Payment* | 29969 | 69940 |
| *Stock Level* | 2987 | 6879 |
| *Výsledok* | *5724,888* | *12959,33* |

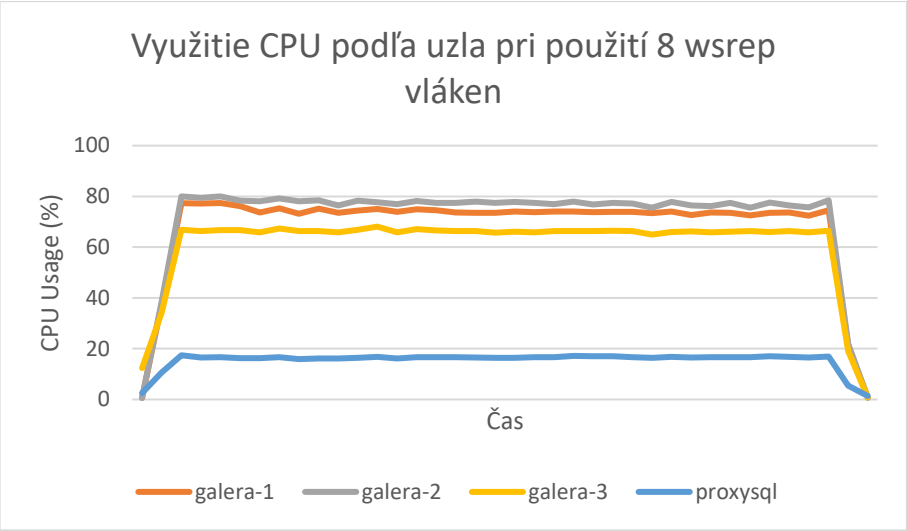
Zdroj: vlastné spracovanie

*Graf 26: Vplyv 8 wsrep vláken na rýchlosť*


Zdroj: vlastné spracovanie

Táto optimalizácia nám priniesla najväčšie zisky vo výkone distribuovaného systému a to až o 126%.

*Graf 27: Využitie CPU podľa uzla pri použití 8 wsrep vláken*




Zdroj: vlastné spracovanie

Okrem zvýšeného výkonu sme mohli pozorovať nárast využitia CPU z približne 30 % na väčšine uzlov až do 60 až 80%. Aj napriek tomu, že optimalizácia zdrojov systému nebola naším primárnym cieľom, optimalizáciu zdrojov sme považovali za pozitívny prínos. Nastavenie *wsrep* vláken na hodnote 8 sme ponechali do ďalších krokov optimalizačného procesu.

### *4.2.6 Časovanie ukladania záznamov na disk*

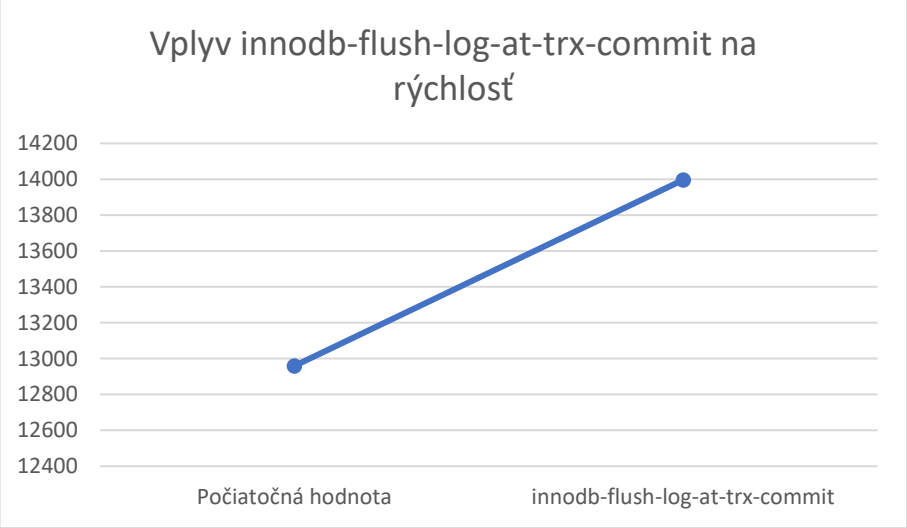
Následne sme sa pokúsili optimalizovať časové rozostupy zápisu dát databázového systému na disk. Toto nastavenie by sme vzhľadom možnosť straty dát pri páde databázového systému neodporúčali implementovať na databázové systémy, ktoré nie sú distribuované a majú iba jeden uzol. Avšak riziko straty dát je v prípade distribuovaného systému eliminované tým, že sa dáta súčasne nachádzajú na viacej ako jednom uzle.

Tabuľka 14: Vplyv innodb-flush-log-at-trx-commit na rýchlosť

| *Druh testu* | *Počiatočné hodnoty* | *innodb-flush-log-at-trx-commit=2* |
|---|---|---|
| *Delivery* | 6669 | 7125 |
| *New Order* | 75357 | 81109 |
| *Order Status* | 6747 | 7267 |
| *Payment* | 69940 | 75833 |
| *Stock Level* | 6879 | 7482 |
| *Výsledok* | *12959,33* | *13996,44* |

Zdroj: vlastné spracovanie

Graf 28: Vplyv innodb-flush-log-at-trx-commit na rýchlosť




Zdroj: vlastné spracovanie

Vo výsledku sme mohli pozorovať zrýchlenie systému pri nastavení zápisov na disk každú sekundu o 8% oproti predchádzajúcej iterácii databázového systému. Nakoľko negatívne aspekty tohto nastavenia boli eliminované v distribuovanom systéme, ponechali sme ho do poslednej optimalizačnej fázy.

### *4.2.7 InnoDB záznamové súbory*

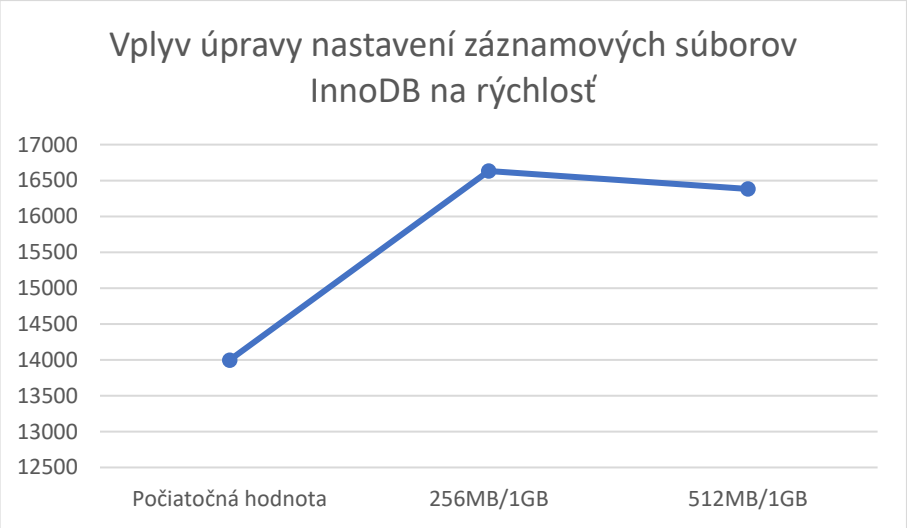
V poslednej časti optimalizácie distribuovaného databázového systému sme sa pozreli na vplyv zmien veľkosti záznamových súborov *InnoDB* na disku a v pamäti RAM. Testovali sme dve kombinácie hodnôt a to 256MB v pamäti RAM / 1GB na disku a 512MB v pamäti RAM a 1GB na disku.

*Tabuľka 15: Vplyv zmien záznamových súborov InnoDB na rýchlosť*

| | | *innodb_log_buffer_size / innodb_log_file_size* | |
|---|---|---|---|
| *Druh testu* | Počiatočné hodnoty | 256MB/1GB | 512MB/1GB |
| *Delivery* | 7125 | 8375 | 8532 |
| *New Order* | 81109 | 96726 | 95081 |
| *Order Status* | 7267 | 8734 | 8562 |
| *Payment* | 75833 | 89756 | 88600 |
| *Stock Level* | 7482 | 8828 | 8592 |
| *Výsledok* | *13996,44* | *16631,85* | *16382,38* |

Zdroj: vlastné spracovanie

*Graf 29: Vplyv zmien záznamových súborov InnoDB na rýchlosť*




Zdroj: vlastné spracovanie

Po vykonaní testovania sme mohli pozorovať zrýchlenie o približne 18% pri nastavení zápisu 256MB záznamov do pamäte RAM a 1GB záznamov na disk. Navýšenie pamäte RAM na väčšiu hodnotu už nemalo pozitívny vplyv na rast výkonu a vzhľadom na to sme ostali na hodnote 256MB.

## 4.3 Výsledky optimalizácie distribuovaného databázového systému

V predchádzajúcich kapitolách sme sa pozreli na rôzne spôsoby ako čo najefektívnejšie optimalizovať distribuovaný databázový systém MySQL Galera Cluster. Táto časť práce slúžila ako sumár a konečné porovnanie výsledkov. Nižšie môžeme nájsť tabuľku, v ktorej

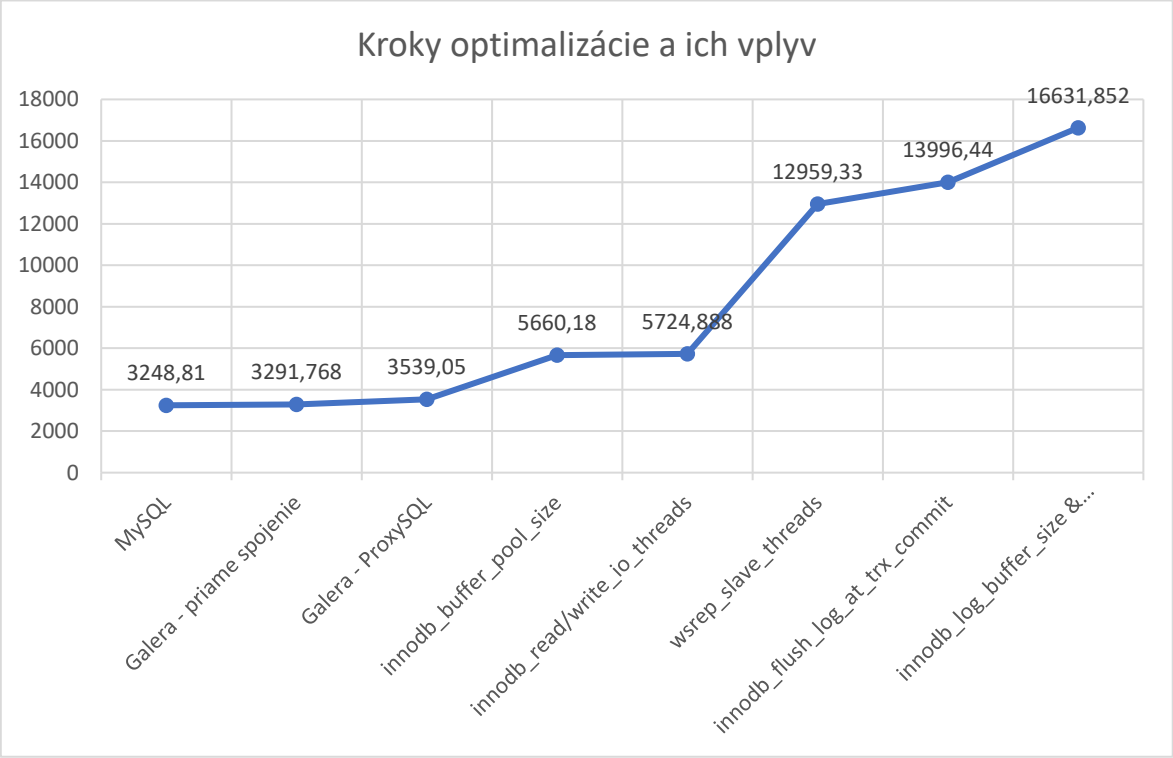
sú zobrazené kroky optimalizácie, ktoré viedli ku zlepšeniu výkonu systému. Hodnoty výsledkov boli zaokrúhlené na jedno desatinné číslo.

*Tabuľka 16: Výsledky optimalizácie*

| | *MySQL* | *Galera - priame spojenie* | *Galera - ProxySQL* | *innodb_buffer_pool_size* | *innodb_read/write_io_threads* | *wsrep_slave_threads* | *innodb_flush_log_at_trx_commit* | *innodb_log_buffer_size & innodb_log_file_size* |
|---|---|---|---|---|---|---|---|---|
| *Výsledok* | 3248,8 | 3291,7 | 3539,0 | 5660,2 | 5724,9 | 12959,3 | 13996,4 | 16631,9 |
| *% rozdiel* | | | +7,5% | +60% | +1,1% | +126,3% | +8% | +18,8% |
| *Celkový rozdiel oproti Galera clusteru v základnej konfigurácii* | | | | | | | | ***+405,2%*** |

Zdroj: vlastné spracovanie

*Graf 30: Kroky optimalizácie a ich vplyv*




Zdroj: vlastné spracovanie

V porovnaní s pôvodným konfiguračným súborom boli pridané nasledovné hodnoty na všetky uzly databázového systému:

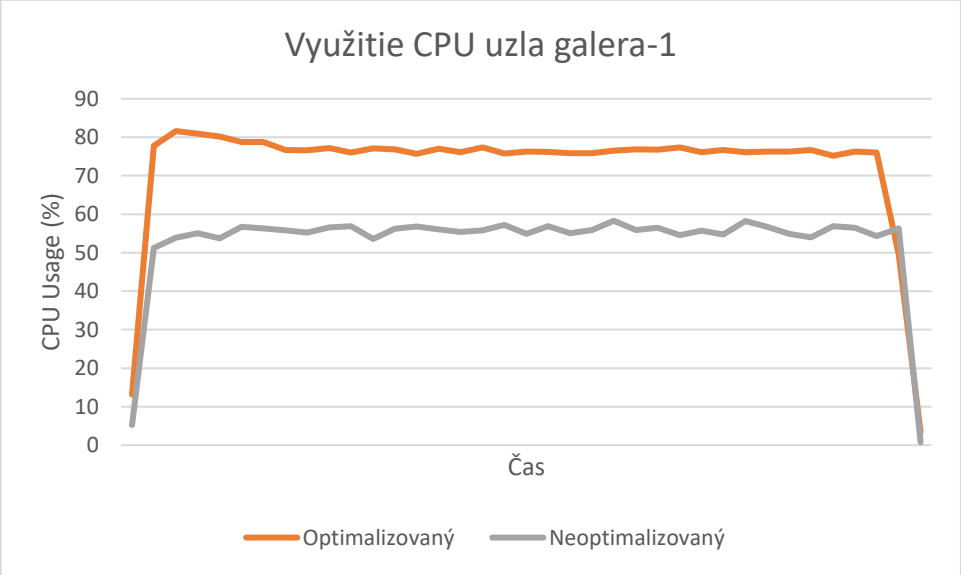
```
innodb_buffer_pool_size = 34359738368
innodb_read_io_threads = 8
innodb_write_io_threads = 4
wsrep_slave_threads = 8
innodb_flush_log_at_trx_commit = 2
innodb_log_buffer_size = 256M
innodb_log_file_size = 1G
```

Ako sme mohli vidieť z predchádzajúcej tabuľky a grafu, najväčší dopad na optimalizáciu databázového systému malo nastavenie replikácie premennou *wsrep_slave_threads*. Za zmienenie taktiež stálo nastavenie pamäte RAM, ktorej optimalizácia zvýšila výkon systému o 60%.

Na druhú stranu, nastavenie *innodb_read_threads* a *innodb_write_threads* neprinieslo skoro žiadny prínos. Prínos 1% výkonu sme mohli považovať za štatistickú odchýlku. Taktiež sme sa stretli s nastaveniami, ktoré bolo vhodnejšie ponechať na pôvodných hodnotách, nakoľko prispeli ku zníženiu výkonnosti databázového systému.

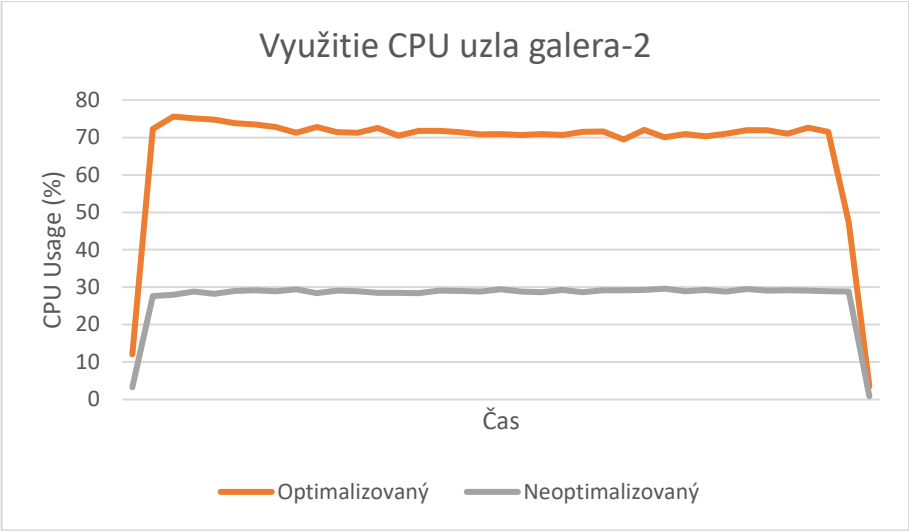
Celkovo sa nám však podarilo distribuovaný databázový systém optimalizovať tak, že oproti pôvodnej verzii bol rýchlejší o 402%, čo sme považovali za úspešnú optimalizáciu.

*Graf 31: Využitie CPU uzla galera-1 na začiatku a na konci optimalizácie*



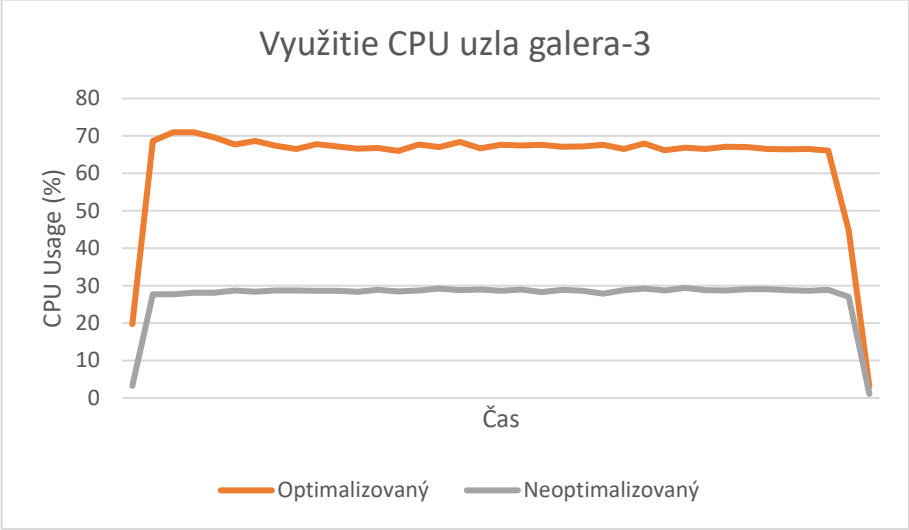

Zdroj: vlastné spracovanie

*Graf 32: Využitie CPU uzla galera-2 na začiatku a na konci optimalizácie*




Zdroj: vlastné spracovanie

*Graf 33: Využitie CPU uzla galera-3 na začiatku a na konci optimalizácie*




Zdroj: vlastné spracovanie

Na záver sme ešte uviedli využitie zdrojov CPU pre jednotlivé uzly systému. Ako sme spomínali v teoretickej časti, využitie CPU by nemalo pre optimálnu funkčnosť presahovať 80% ale zároveň by nemalo byť nedostatočne využité. V prípade nášho systému sa nám podarilo zvýšiť využitie z 30 až 60% CPU (využitie 60% CPU bolo iba na jednom uzli) na 70 až 80%. Teda napriek tomu, že naším cieľom v tejto práci nebola optimalizácia využitia zdrojov CPU, môžeme povedať, že distribuovaný systém bol vhodne navrhnutý aj z pohľadu využitia systémových zdrojov.

## 4.4 Ďalšie odporúčania

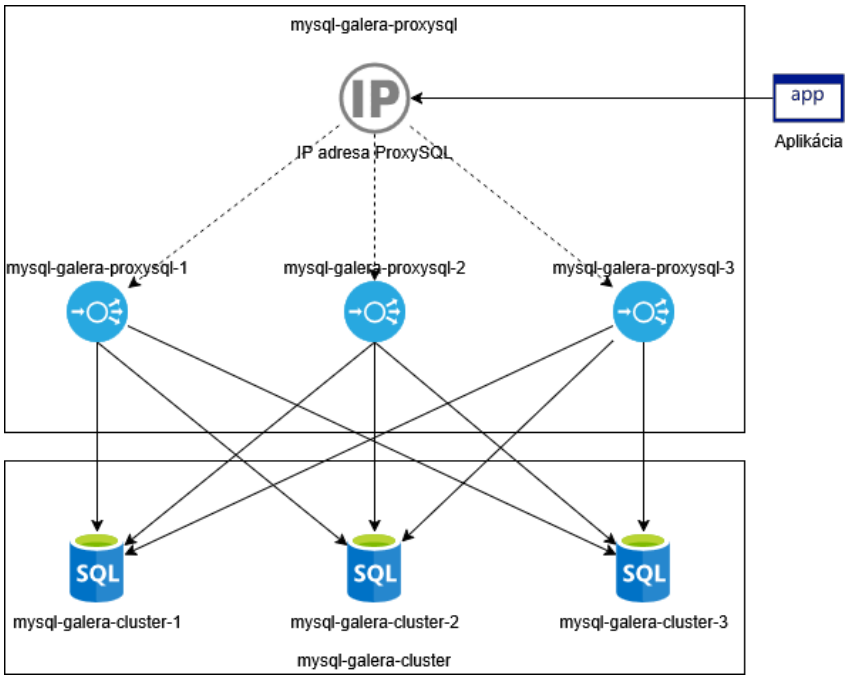
V tejto kapitole sme sa venovali ďalším optimalizačným krokom, ktoré by bolo možné vykonať na distribuovanom databázovom systéme v prípade, že by podnik naďalej rástol a navrhnutý databázový systém by už nepostačoval jeho požiadavkám. Táto časť sa nezameriavala iba na optimalizáciu rýchlosti systému ale aj na jeho zabezpečenie, resp. redundanciu častí a odporúčala vykonanie dvoch krokov, ktoré spolu úzko súvisia.

Prvým krokom bolo pridanie viacero uzlov do systému. Tieto uzly by boli umiestnené tak aby sa nachádzali na troch geograficky rozličných miestach (napríklad v troch rôznych skladoch spoločnosti). Autori práce odporúčali minimálne päť až sedem uzlov, nakoľko pre správne fungovanie distribuovaného systému je nutné mať v prevádzke aspoň tri uzly, čo by umožnilo stratu dvoch až štyroch rôznych uzlov. Tento úkon by síce zvýšil latenciu databázového systému, avšak benefitom by bola zvýšená dostupnosť.

Druhým krokom, ktorý úzko súvisí s pridaním viacerých uzlov bolo vyhradenie dvoch alebo viacero uzlov iba na operácie čítania. Tým by boli odľahčené uzly, ktoré by sa mohli venovať iba vykonávaniu jednej operácie – čítaniu alebo zápisu. Zároveň v prípade potreby, napríklad z dôvodu výpadku uzlov povoľujúcich zápis by bolo jednoduché upraviť konfiguráciu *ProxySQL* tak aby umožnila dočasný zápis dát aj na uzly určené iba pre čítanie bez toho aby bola ohrozená funkčnosť databázového systému.

Taktiež by bolo možné vytvoriť viacero ProxySQL serverov, nakoľko v navrhovanom rozdelení je ProxySQL server jediným miestom, pri ktorom nebola zabezpečená vysoká dostupnosť. Tieto servery by mohli mať za pomoci nástroja ako *keepalived* zdieľanú virtuálnu IP, ktorá by zjednodušila nastavovanie sieťových spojení jednotlivých databázových serverov.

*Diagram 6: Návrh vysoko dostupného systému*




Zdroj: vlastné spracovanie

Vzhľadom na nedostatok hardvérových zdrojov a na fakt, že tieto úpravy sa týkajú predovšetkým zlepšenia dostupnosti databázového systému a nie zvýšenia jeho rýchlosti, neboli tieto zmeny implementované ako súčasť tejto práce.

# Záver

Naprieč touto prácou sme sa zamerali na optimalizáciu distribuovaného databázového systému *MySQL Galera Cluster*. Naším cieľom bolo navrhnúť distribuovaný databázový systém a následne ho optimalizovať za cieľom získania čo najväčšieho výkonu pre malý až stredný podnik.

Pred začiatkom testovania však bolo nutné vysvetliť pojmy, s ktorými sme sa počas testovania stretli. Predstavili sme si rôzne databázové systémy ale aj iné technológie ktoré bolo možné využiť pri optimalizácii distribuovaného databázového systému ako napríklad *ProxySQL*. Tiež sme si predstavili nástroje, ktoré sme používali ale nesúviseli priamo s databázovými systémami. Taktiež sme sa zamerali na predstavenie a úvod do rôznych premenných databázového systému *MySQL*. Tieto premenné sme v neskoršej časti práce používali na optimalizáciu.

V metodickej časti práce sme navrhli spôsob, ktorým boli jednotlivé zmeny distribuovaného databázového systému merané po výkonnostnej stránke. Na meranie sme používali nástroj *DBT2*. Tento nástroj nám ponúkal väčšie množstvo testovacích situácii a každá z nich vykonávala špecifický typ záťažového testu. Okrem merania výsledkov sme si predstavili aj základný návrh distribuovaného databázového systému, spolu s parametrami jednotlivých serverov, ktorý sme naprieč prácou zdokonaľovali a predstavili dôvody pre ktoré sme sa rozhodli využívať špecifickú technológiu na vytvorenie distribuovaného databázového systému. Tiež sme pozreli na to, aké možnosti nám ponúkajú operačné systémy založené na *GNU/Linux* pre simuláciu komplikácií na sieťovom spojení.

Po teoretickom vysvetlení fungovaní databázových systémov a vysvetlení metodiky optimalizácie distribuovaného databázového systému sme sa začali venovať samotnej optimalizácii systému. Na úvod sme vykonali porovnanie medzi nedistribuovaným systémom a distribuovaným systémom. Na tento systém sme následne nasadili aplikáciu na rozdistribuovanie záťaže naprieč všetkými uzlami. Po rozdistribuovaní záťaže sme sa pozreli na nastavenie rôznych premenných. Pri niektorých premenných sme mohli pozorovať štatisticky nevýznamné zmeny vo výkone, pri iných sme sa stretli s horšením výkonu distribuovaného systému. Zmeny ktoré nám priniesli negatívny dopad sme nezakomponovávali do ďalších krokov ale boli vynechané. Taktiež sme identifikovali niektoré premenné, po ktorých úprave sa nám podarilo výrazne ovplyvniť rýchlosť databázového systému. Okrem optimalizácie sme sa pozreli aj na vplyv sieťového spojenia

na výkon distribuovaného systému. Konkrétne sme sa zamerali na stratu paketov a vysokú latenciu v sieťovom spojení. V posledných častiach sme vyhodnotili všetky naše zmeny a predstavili výsledok tejto práce. Výsledkom bol distribuovaný databázový systém, ktorý bol v porovnaní s databázovým systémom navrhnutým na začiatku tejto práce o viac ako 400% výkonnejší.

Na záver sme sa ponúkli dodatočné odporúčania pre navrhnutý systém do budúcna. Tieto odporúčania sa už do veľkej miery netýkali priamo optimalizácie za účelom zvýšenia výkonu systému ale išlo o zmeny, ktoré by bolo vhodné implementovať pri budúcom raste alebo nasadení distribuovaného systému do produkčného prostredia. Išlo primárne o návrhy ktoré by zabezpečili vysokú dostupnosť všetkých súčastí systému.

# Zoznam použitej literatúry


**45 Drives Ltd. n. d..** ZFS Caching. *45Drives.* [Online] n. d. [Dátum: 11. Február 2025.] https://www.45drives.com/community/articles/zfs-caching/

**Benny, Ou. 2017.** CPU Usage vs Load. *Medium.* [Online] 17. Jún 2017. [Dátum: 3. Marec 2025.] https://estl.tech/cpu-usage-vs-load-ecca22287b21

**Blåudd, Magnus. 2024.** MySQL NDB Cluster replication: Introduction. *The Oracle MySQL Blog.* [Online] 4. Apríl 2024. [Dátum: 10. Február 2025.] https://blogs.oracle.com/mysql/post/introduction-to-mysql-ndb-cluster-replication

**Codership Ltd. n. d..** Replication API. *Galera Cluster.* [Online] n. d. [Dátum: 14. Marec 2025.] https://galeracluster.com/library/documentation/architecture.html

**Codership Oy. 2025.** Galera Cluster Documentation. [Online] 25. Február 2025. [Dátum: 10. Marec 2025.] https://galeracluster.com/library/galera-documentation.pdf

**Davies, Alex a Fisk, Harrison. 2006.** *MySQL Clustering.* Uppsala : MySql Press, 2006. s. 216. ISBN 0672328550.

**Descamps, Frederic. 2024.** MySQL enters a new decade! *The Oracle MySQL Blog.* [Online] 23. December 2024. [Dátum: 14. Marec 2025.] https://blogs.oracle.com/mysql/post/mysql-enters-a-new-decade

**Drake, Mark. 2021.** What is a Packet? *DigitalOcean Community.* [Online] 12. Január 2021. [Dátum: 13. Marec 2025.] https://www.digitalocean.com/community/tutorials/what-is-a-packet

**Free Software Foundation. 2011.** NetEm. *Bash manual page.* [Online] 25. November 2011. [Dátum: 21. Január 2025.] https://man7.org/linux/man-pages/man8/tc-netem.8.html

**Goodwin, Michael. 2023.** What is latency? *IBM.* [Online] 15. August 2023. [Dátum: 16. Marec 2025.] https://www.ibm.com/think/topics/latency

**HAProxy. n. d..** Description. *HAProxy.* [Online] n. d. [Dátum: 13. Marec 2025.] https://www.haproxy.org/

**Harford, Tim. 2017.** How the world's first accountants counted on cuneiform. *BBC News.* [Online] 12. Jún 2017. [Dátum: 10. Február 2025.] https://www.bbc.com/news/business-39870485.

**IONOS editorial team. 2023.** InfluxDB. *IONOS Digital Guide.* [Online] 5. December 2023. [Dátum: 14. Marec 2025.] https://www.ionos.com/digitalguide/hosting/technical-matters/what-is-influxdb/

**IR Team. n. d..** Network Latency - Common Causes and Best Solutions. [Online] n. d. [Dátum: 16. Marec 2025.] https://www.ir.com/guides/what-is-network-latency

**Jameson, Brooke. 2024.** What is a Network Packet? *Netscout Learning Center.* [Online] 23. Január 2024. [Dátum: 16. Marec 2025.] https://www.netscout.com/what-is/packet

**Kanyi, Stephen. 2018.** MySQL Load Balancing with HAProxy. *Medium.* [Online] 8. Február 2018. [Dátum: 16. Marec 2025.] https://medium.com/@StephenKanyiW/mysql-load-balancing-with-haproxy-46b25b0295dd

**Kusnetzky, Dan. 2017.** A Brief History of Clustering. *Virtualization & Cloud Review.* [Online] 31. Január 2017. [Dátum: 10. Február 2025.] https://virtualizationreview.com/articles/2017/01/31/a-brief-history-of-clustering.aspx

**MariaDB Foundation. n. d..** MariaDB in brief. *MariaDB Foundation.* [Online] n. d. [Dátum: 14. Marec 2025.] https://mariadb.org/en/

**MariaDB. 2025.** MariaDB versus MySQL: Compatibility. *MariaDB Knowledge Base.* [Online] 13. Január 2025. [Dátum: 24. Marec 2025.] https://mariadb.com/kb/en/mariadb-vs-mysql-compatibility/#incompatibilities-between-mariadb-107-and-mysql-80

**Mucci, Tim. 2024.** What is structured query language (SQL)? *IBM.* [Online] 31. Máj 2024. [Dátum: 10. Február 2025.] https://www.ibm.com/think/topics/structured-query-language

**Open Source Development Labs, Inc. n. d..** Database Test 2 (DBT-2) Documentation. *Database Test 2 (DBT-2) Documentation.* [Online] n. d. [Dátum: 14. Január 2025.] https://osdldbt.github.io/dbt2/#introduction

**Oracle Corporation. n. d.. (a)** Chapter 19 Replication. *MySQL 8.3 Reference Manual.* [Online] n. d. [Dátum: 24. Február 2025.] https://dev.mysql.com/doc/refman/8.3/en/replication.html

**Oracle Corporation. 2025.** InnoDB Startup Options and System Variables. *MySQL 8.4 Reference Manual.* [Online] 2025. [Dátum: 4. April 2025.] https://dev.mysql.com/doc/refman/8.4/en/innodb-parameters.html

**Oracle Corporation. n. d.. (b)** NDB and InnoDB Feature Usage Summary. *MySQL NDB Cluster 8.0.* [Online] n. d. [Dátum: 3. Marec 2025.] https://dev.mysql.com/doc/mysql-cluster-excerpt/8.0/en/mysql-cluster-ndb-innodb-usage.html

**Pachev, Sasha. 2007.** *Understanding MySQL Internals: Discovering and Improving a Great Database.* Sebastopol : O'Reilly Media, Inc., 2007. s. 258. ISBN 9780596009571.

**Pandora FMS team. 2025.** Packet Loss in Networks: Diagnosis, Causes and Solutions. [Online] 5. Február 2025. [Dátum: 16. Marec 2025.] https://pandorafms.com/blog/packet-loss/

**Pattanayak, Nanda Gopal. 2022.** Setting up a basic GTID based MySQL Replication setup. *Medium.* [Online] 5. September 2022. [Dátum: 10. Marec 2025.] https://medium.com/@nandagopal05/setting-up-a-basic-gtid-based-mysql-replication-setup-e615890672

**Proxmox Server Solutions GmbH. n. d..** Features. *Proxmox.* [Online] n. d. [Dátum: 10. Jaunár 2025.] https://www.proxmox.com/en/products/proxmox-virtual-environment/features

**PubNub Labs Team. 2024.** What is Network Packet Loss. [Online] 20. November 2024. [Dátum: 16. Marec 2025.] https://www.pubnub.com/blog/network-packet-loss/

**Raab, Francois, Kohler, Walt a Shah, Amitabh. n. d..** Overview of the TPC-C Benchmark - The Order-Entry Benchmark. *TPC.* [Online] n. d. [Dátum: 6. Jaunár 2025.] https://www.tpc.org/tpcc/detail5.asp

**Razzoli, Federico. 2014.** *Mastering MariaDB.* Birmingham : Packt Publishing Ltd., 2014. s. 350. ISBN 978-1-78398-154-0.

**Risal, Shashwot. 2022.** Load Balancing MySQL servers using ProxySQL. *Medium.* [Online] 25. Január 2022. [Dátum: 15. Marec 2025.] https://shashwotrisal.medium.com/load-balancing-mysql-servers-using-proxysql-ff5ab6d1d4ea

**Samuel, Nicholas. 2025.** Using MySQL BinLogs: A Detailed Guide. *Hevo.* [Online] 2. April 2025. [Dátum: 4. April 2025.] https://hevodata.com/learn/using-mysql-binlog/

**ScaleGrid Inc. 2019.** 2019 Database Trends – SQL™ vs. NoSQL, Top Databases, Single vs. Multiple Database Use. *ScaleGrid.* [Online] 4. Marec 2019. [Dátum: 10. Február 2025.] https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/

**SentinelOne. 2021.** Everything You Need to Know About Linux CPU Load. *SentinelOne Blog.* [Online] 19. Máj 2021. [Dátum: 5. Marec 2025.] https://www.sentinelone.com/blog/linux-cpu-load/

**Simic, Sofija. 2022.** What is a Hypervisor? Types of Hypervisors 1 & 2. *PhoenixNAP.* [Online] 29. September 2022. [Dátum: 4. Marec 2025.] https://phoenixnap.com/kb/what-is-hypervisor-type-1-2

**SRTLab. 2023.** Using NetEm to Emulate Networks. *SRT CookBook.* [Online] 20. Jul 2023. [Dátum: 16. Január 2025.] https://srtlab.github.io/srt-cookbook/how-to-articles/using-netem-to-emulate-networks.html

**Stack Exchange Inc. n. d..** StackOverflow Developer Survey 2024. *StackOverflow.* [Online] n. d. [Dátum: 25. Marec 2025.] https://survey.stackoverflow.co/2024/technology#most-popular-technologies-database-prof

**Tomer, Ben David. 2024.** Understanding Load Average vs. CPU Utilization. *Medium.* [Online] 1. Október 2024. [Dátum: 3. Marec 2025.] https://medium.com/@Tom1212121/understanding-load-average-vs-cpu-utilization-46f1f3d40ee2

**Tusa, Marco. 2023.** Comparisons of Proxies for MySQL. *Percona.* [Online] 20. Marec 2023. [Dátum: 11. Marec 2025.] https://www.percona.com/blog/comparisons-of-proxies-for-mysql/

**Tusa, Marco. 2024.** Is MySQL Router 8.2 Any Better? *Percona.* [Online] 11. Január 2024. [Dátum: 13. Marec 2025.] https://www.percona.com/blog/is-mysql-router-8-2-any-better/

**Vierling, Giovanni. 2023.** The database evolution: modernization for a data-driven world. *Compact_ - powered by KPMG Netherlands.* [Online] Október 2023. [Dátum: 10. Február 2025.] https://www.compact.nl/articles/the-database-evolution-modernization-for-a-data-driven-world/

**Wojsław, Damian. 2017.** *Introducing ZFS on Linux: Understand the Basics of Storage with ZFS.* Szczecin : Springer Science+Business Media New York, 2017. s. 122. ISBN 978-1-4842-3306-1