VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza výkonnosti otevřených messaging systémů

Perfromance Analysis of Open Source Messaging Systems

2012 Lukáš Sojka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Sojka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Analýza výkonnosti otevřených messaging systémů
Perfromance Analysis of Open Source Messaging Systems

Zásady pro vypracování:

Cílem práce je analyzovat nejrozšířenější stávající otevřené messaging systémy, popsat je, a v praktických experimentech ověřit jejich výkonnost při přenosu velkého množství zpráv.

Implementovaný systém bude simulovat nasazení messaging systémů jako komunikační páteře distribuované aplikace s velkým provozem přenášených zpráv o velikosti v řádu až stovek kB (např. elektronické dokumenty). V rámci práce bude zhodnocena výkonnost takového řešení, udržovatelnost, škálovatelnost, přenositelnost, a jednoduchost integrace s jinými SW systémy.

Osnova:
1. Seznámení s MOM, ESB, systémy pro asynchronní komunikaci.
2. Přehled nejznámějších komerčních a otevřených implementací, příklady použití.
3. Návrh a implementace testovací aplikace.
4. Provedení testů s různými implementacemi messaging systémů.
5. Zhodnocení výsledků.

Seznam doporučené odborné literatury:

G. Alonso, F. Casati, H. Kuno, V. Machiraju: Web Services: Concepts, Architectures and Applications (Data-Centric Systems and Applications), Springer 2010

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Krömer, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012

Luboš Cajthaml

.....

Chtěl bych poděkovat svému vedoucímu Ing. Pavlu Krömerovi, Ph.D. za odborné vedení mé práce, rady, podměty a čas, který mi věnoval.

Abstrakt

Tato diplomová práce se zaobírá tématikou messaging systémů – Message-Oriented Middlewaru (MOM), který dnes tvoří důležitou součást v oblasti systémové integrace. Tato práce si klade za cíl seznámit se základními principy messagingu, přiblížit nejznámější komerční a otevřené implementace MOM a především navrhnout a implementovat testovací aplikaci, která by využívala MOM jako komunikační páteř pro přenos velkého množství dat o velkých velikostech. Dále pak seznámit s existujícími výkonnostními porovnáními a především navrhnout testovací případy a provést výkonnostní testy na vytvořené testovací aplikaci pro vybrané implementace MOM. Nakonec pak provést analýzu a vyhodnocení získaných výsledků.

Klíčová slova: Message-Oriented Middleware, MOM, Java Message Service, JMS, messaging, messaging systém, analýza výkonnosti

Abstract

This Diploma thesis describes theme Messaging Systems – Message-Oriented Middleware (MOM), which today forms an important part in system integration. This thesis behind aim acquaint with basic principles of messaging, bring the most popular commercial and open source implementation of the MOM closer and especially design and implement test application. This application uses the MOM as a communications backbone for the transmission of large amounts of large size data. Further acquaint with existing performance comparisons and especially design test cases and perform the performance tests on test application for selected implementations of MOM. And the finally analyze and evaluate obtained results.

Keywords: Message-Oriented Middleware, MOM, Java Message Service, JMS, messaging, messaging system, performance analysis

Seznam použitých zkratek a symbolů

AMQP	–	Advanced Message Queuing Protocol
API	–	Application Programming Interface
ESB	–	Enterprise Service Bus
HTML	–	Hyper Text Markup Language
IM	–	Instant Messaging
JAAS	–	Java Authentication and Authorization Service
JDBC	–	Java Database Connectivity
JMS	–	Java Message Service
JMX	–	Java Management Extensions
JNDI	–	Java Naming and Directory Interface
JORAM	–	Java Open Reliable Asynchronous Messaging
LDAP	–	Lightweight Directory Access Protocol
MOM	–	Message-Oriented Middleware
MS	–	Messaging System, Messaging systém
RPC	–	Remote Procedure Call
SOA	–	Service Oriented Architecture
SOAP	–	Simple Object Access Protocol
STOMP	–	Streaming Text Oriented Message Protocol
SSL	–	Secure Sockets Layer
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
URL	–	Uniform Resource Locator
WS	–	Web Service, Webová služba
WSDL	–	Web Services Description Language
XML	–	Extensible Markup Language
XMMP	–	Extensible Messaging and Presence Protocol

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

V dnešní době se stále více stává běžné, že každá firma využívá pro podporu svých podnikových procesů více než jednu aplikaci či informační systém. Typické je, že firmy používají různé specializované aplikace různých výrobců, které jsou postaveny na rozličných platformách či operačních systémech. S tímto rostoucím počtem heterogenních aplikací a zvyšujícími se požadavky na dostupnost, aktuálnost a vzájemné sdílení dat či zvýšení efektivity se objevují stále větší nároky na vzájemné propojování těchto heterogenních aplikací a jejich vzájemnou integraci. Těmito problémy se zabývá oblast systémové integrace.

Tato diplomová práce se věnuje oblasti komunikace aplikací založená na posílání zpráv – messaging systémům (Message-Oriented Middleware, MOM), které dnes tvoří důležitou součást právě v oblasti systémové integrace a tvoří jádro dnes asi nejperspektivnějšího řešení v rámci integrace aplikací tj. podniková sběrnice služeb (Enterprise Service Bus, ESB).

Uvedení messagingu do širšího kontextu systémové integrace, jeho základní principy a vlastnosti, integrace s webovými službami a Enterprise Service Bus jsou popsány v druhé a třetí kapitole této práce. Zároveň je zde popsána JMS specifikace, která umožňuje aplikacím napsaných v Javě jednotným způsobem komunikovat s různými implementacemi MOM. Ve čtvrté kapitole pak jsou popsány nejznámější komerční a otevřené implementace MOM, jejich vlastnosti a použití v dalších systémech.

Pátá až sedmá kapitola je věnována vyvinuté testovací aplikaci a JMS Adaptéru. Zde jsem popsal motivaci pro vznik aplikace, její princip a realizované funkce i její vztah k MOM a JMS specifikaci a dále také návrh aplikace a použité technologie. Nakonec je také popsán implementovaný JMS Adaptér, který poskytuje skoro jednotný způsob práce s vybranými implementacemi MOM.

V osmé kapitole popisuji nasazování jednotlivých vybraných implementací MOM, které budou analyzovány. Těmito implementacemi jsou ActiveMQ, HornetQ, OpenMQ, JORAM a OpenJMS.

Největší část práce (9. a 10. kapitola) je pak věnována samotné analýze jednotlivých implementací messaging systémů. Jsou zde zmíněny existující výkonnostní porovnání, následované návrhem a popisem realizace výkonnostních testů, analýzou a zhodnocením výsledků těchto testů. V úplném závěru je pak celkové zhodnocení testovaných messaging systémů a to především z hlediska výkonnostního, ale i z hlediska stability, jednoduchosti integrace s jinými SW systémy, přenositelnosti, udržovatelnosti, aj.

2 Komunikace aplikací založená na posílání zpráv

2.1 Úvod

Téma této diplomové práce se zaměřuje na oblast *messaging systémů (Message-Oriented Middleware).* Oblast messaging systémů velice úzce souvisí s pojmy jako je *systémová integrace*, *vzájemné propojování heterogenních systémů*, případně *subsystémů*, které jsou v dnešní době stále častěji skloňovány. Proto by bylo vhodné na úvod této diplomové práce stručně zmínit hlavní *integrační styly* (způsoby) pro vzájemné propojování aplikací a tak umístit pojem *messaging systém* do širšího kontextu. Následující informace byly čerpány z [1, 2, 3].
Hlavní integrační styly jsou:

1. *Přenos údajů přes sdílené soubory (File Transfer)* – aplikace vytvářejí soubory s údaji, které mohou zpracovávat jiné aplikace a tak si navzájem vyměňovat informace. Jedná se o nejjednodušší řešení propojení aplikací (middlewaru). *Výhodou* tohoto řešení je jeho jednoduchost a univerzálnost (snad každý programovací jazyk a operační systém umí pracovat se soubory, atd.). *Nevýhodou* jsou problémy se synchronizací a obtížným monitorováním.

2. *Sdílená databáze (Shared Database)* – aplikace ukládají své data do sdílené databáze. *Výhody* tohoto řešení jsou: k dispozici jsou vždy aktuální data, snadný přístup (SQL). *Nevýhody pak:* v případě integrace další aplikace musí dojít k její úpravě tak, aby mohla využívat sdílenou databázi (což není vždy možné), výrobci některých aplikací si vyhrazují právo na změnu struktury databáze.

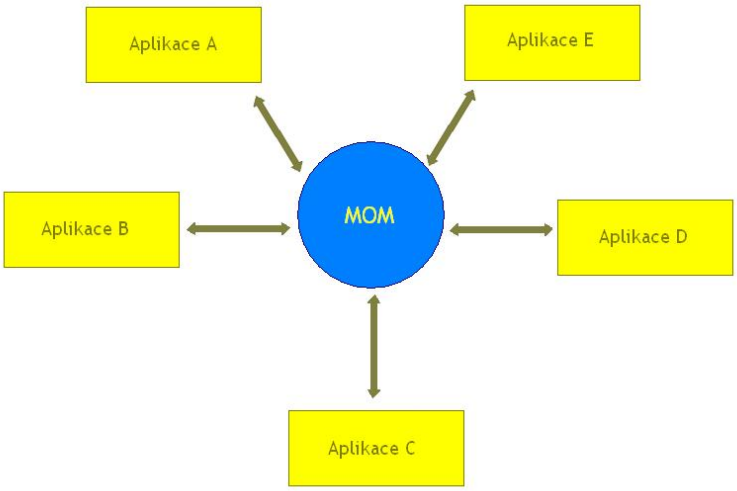
3. *Volání vzdálených procedur (Remote Procedure Call)* – aplikace si předávají data tak, že jedna aplikace nabízí své procedury. Ostatní aplikace mohou tyto vzdálené procedury využívat, jako kdyby byly lokální. Procedura je „černá skříňka“, u které nás zajímá jen její rozhraní a výsledek, který vyprodukuje, ne jak je implementována, což je *výhoda. Nevýhodami jsou:* velké provázání aplikací, klient musí čekat, až mu server vrátí odpověď, případné změny rozhraní, či nedostupnost aplikace s volanou procedurou.

4. *Posílání zpráv (Messaging, Message-Oriented Middleware)* – aplikace si vyměňují data pomocí posílání zpráv přes *systém pro doručování zpráv (messaging sytém).* Této metodě se detailně věnuje celá následující kapitola.

Kromě integračních stylů je také vhodné uvést dva základní **interakční modely:**

1. *Synchronní komunikace* – v případě synchronní komunikace je „volající“ aplikace zablokována a čeká, dokud není volaný kód proveden a je jí vrácena odpověď. „Volající aplikace“ nemá kontrolu nad řízením. Musí spoléhat na navrácení řízení z „volané“ aplikace. Typickým příkladem synchronní komunikace je *Remote Procedure Call (RPC).*

2. *Asynchronní komunikace* – v případě asynchronní komunikace je „volajícímu“ ponecháno řízení. Nemusí tedy čekat na odpověď. Požadavek, který „volající“ ode-



Obrázek 1: Topologie distribuované aplikace s MOM

slal, nemusí být hned vykonán. Typickým příkladem asynchronní komunikace je *Message-Oriented Middleware (MOM).*

Další informace o této problematice je možné nalézt na [4].

2.2 Message-Oriented Middleware

Následující informace byly čerpány především z [2, 3, 5].

Message-Oriented Middleware (MOM) je softwarová a hardwarová platforma zajišťující komunikaci mezi aplikacemi, která je založená na *asynchronním posílání zpráv.* Její historie sahá až k roku 1980. MOM umožňuje aplikacím (či jejím částem) vzájemnou komunikaci skrze *heterogenní prostředí* a *různé síťové protokoly* a tak značně ulehčuje námahu při integraci rozsáhlých systémů.

Pro zajištění asynchronní povahy přenosu jsou zprávy ukládány typicky do *front (quueu)*, které jsou uchovávány na centrálním serveru – *systému pro doručování zpráv (messaging systém, provider, broker).* Jako analogii k posílání zpráv (MOM) si lze představit velmi rychlou poštovní službu, které je možné předat naší zprávu, a samotné doručení je již pak v zodpovědnosti samotné poštovní služby, která zprávu doručí příjemci, až bude k dispozici. Opakem k MOM je RPC, kde RPC si lze představit jako telefonní hovor mezi dvěma uživateli, kdy každý musí být v konkrétní okamžik přítomen [2]. Výhodou MOM platformy je také, že v případě změny provedené v jedné části systému, či připojení další části, není prakticky nutné provádět změny v ostatních částech systému.

Typická topologie distribuované aplikace s nasazení MOM je vidět na obrázku 1, kde aplikace mezi sebou navzájem komunikují přes messaging systém.

2.2.1 Vlastnosti MOM

2.2.1.1 Volná vazba (Loose coupling) Jak již bylo zmíněno, MOM vkládá mezi odesílatele a příjemce zpráv *další vrstvu – messaging systém (MS)*, kterou odesílatel a příjemce využívá jako nezávislého prostředníka pro výměnu zpráv. Hlavní výhodou tohoto je, že je vytvořená *volná vazba mezi jednotlivými částmi systému.* Je možno velmi snadno připojovat další části systému, aniž by musely být provedeny změny ve stávajících i nově připojovaných částech.

2.2.1.2 Spolehlivost (Reliability) MOM umožňuje zajišťovat *zaručený přenos dat*, což je jednou z jeho hlavních výhod. Toto je zajištěno *uložením dat na serveru konkrétního messaging sytému* a možností *persistence dat*, tj. data přežijí i pád serveru a jsou bezpečně znova načtena z disku. Tato schopnost zaručuje vysokou spolehlivost přenosu a zabraňuje nedoručení zpráv příjemci v případě, že je nedostupný nebo zaneprázdněný a zároveň umožňuje snadný monitoring prováděných operací.

2.2.1.3 Škálovatelnost (Scalability) MOM umožňuje také *oddělení výkonnostních charakteristik jednotlivých subsystémů.* Subsystémy mohou být škálovatelné tak, že jsou na sobě prakticky nezávislé, anebo jen velmi málo. Díky tomu je snadnější se vypořádat s nepředvídatelnými výkyvy v zátěži jednoho subsystému, aniž by došlo k výraznému omezení ostatních.

2.2.1.4 Dostupnost (Availability) MOM poskytuje *vysokou dostupnost*, která je zajištěno volnou vazbou spojení subsystémů, principem asynchronního přenosu a ukládáním dat na serveru MS. Pád jednoho subsystému nevyřadí celý systém, ale zbylá část může bez nebo jen s malým omezením pracovat dál.

Typickým příkladem využití je objednávkový systém a fakturační systém. Objednávkový systém může přijímat objednávky, i když je fakturační systém zrovna mimo provoz. Objednávky jsou bezpečně ukládány na server MS a po znovu zprovoznění fakturačního systému jsou objednávky systémem zpracovány. Zákazník, který systém používá, vůbec nezpozoruje výpadek [5].

2.2.1.5 Rozložení zátěže (Load-balancing) Další výhodou je jednoduchost realizace *load-balancingu*. Ten je snadno realizovatelný díky frontám, v kterých jsou uchovávány zprávy. Pokud nějaký subsystém nestíhá zpracovávat zprávy z fronty, muže být na tuto frontu připojena další instance. Instance pak budou paralelně zpracovávat data.

2.2.1.6 Další vlastnosti MOM Kromě, výše zmíněných, typický vlastností MOM poskytuje ještě další vlastnosti, které jsou ve většině případů podporovány jednotlivými implementace MS.

Tyto vlastnosti jsou:

- *Typ modelu* – základ tvoří dva modely: *Point-to-Point* a *Publish/Subscribe* (budou detailně rozebrány v kapitole 2.3)

- *Transakce a mechanizmy pro zajištění spolehlivosti přenosu* – viz kapitola 2.3.
- *Transformace zpráv* – MS může původní zprávu přijatou od jednoho klienta přetransformovat do tvaru přijatelného pro jiného klienta.
- *Filtrování zpráv* – filtrování umožňuje klientovi si „vytáhnout“ z fronty pouze ty zprávy, které potřebuje.
- *Replikace a přesměrovávání zpráv* – některé MS umožňují replikovat zprávy do více front nebo je přesměrovávat do jiných front.
- *Klastrování* – data MS jsou replikována na více serverech, aby se zabránilo celkovému pádu systému v případě, že by došlo k pádu serveru MS. Dalším důvodem je rozložení zátěže na více serverů v případě potřeby a umožnit další škálování systému.

2.2.2 Protokoly MOM

Pro MOM existuje standard (specifikace) *JMS*, která umožňuje *jednotným způsobem komunikovat mezi klienty* (aplikacemi, subsystémy) napsanými v Javě bez ohledu na konkrétní implementaci MS (více viz kapitola 2.3). Jednou z nevýhod JMS specifikace ovšem je, že nedefinuje protokol přenosu. Proto jednotlivé implementace MS používají různé protokoly. Nejčastěji ovšem je použit *STOMP*, *AMQP*, případně *XMMP protokol* [6, 7].

STOMP – The Streaming Text Oriented Message Protocol je jednoduchý textový protokol navržený pro MOM. Jeho syntaxe je blízka HTTP a je jazykově nezávislý. Tento protokol využívá například ActiveMQ nebo HornetQ [7].

AMQP – The Advanced Message Queuing Protocol druhým a dnes preferovanějším protokolem pro MOM. Jeho hlavním cílem je zajištění spolehlivosti a výkonnosti. AMQP se skládá ze dvou vrstev. Dolní definuje transportní protokol, typicky TPC. Horní vrstva pak definuje konkrétní vlastnosti týkající se MOM (typ výměny zpráv, fronty, atd.). Tento protokol například využívá JORAM, RabbitMQ a mnohé další MS [7].

XMPP – Extensible Messaging and Presence Protocol je protokol převážně určený pro Instant Messaging, ale může být využíván i MOM. Je založený na XML [7].

2.2.3 Výhody a nevýhody MOM

Výhody komunikace aplikací založené na posílání zpráv (MOM) byly z velké části již podrobně popsány výše. Hlavní výhoda vyplývá z *asynchronní povahy messagingu*, kdy není nutno čekat na odpověď od jiné aplikace (subsystému, komponenty). S tím souvisí také *veliká dostupnost systému*. Tím, že MOM vkládá mezi jednotlivé aplikace jakousi komunikační mezivrstvu, je zajištěna vysoká *flexibilita*, *spolehlivost doručení dat*, *rozložení zátěže* a *škálovatelnost*. Navíc je díky tomuto „prostředníku“ možno poskytovat další funkce, jako *transformace*, *filtrování*, *replikace*, *atd.*

Z výše uvedených vlastností MOM vyplývají i jeho nevýhody. Mnohé aplikace potřebují využívat *synchronní způsob komunikace*, asynchronní je pro ně nevyhovující. Na druhou stranu MOM umožňuje určitým způsobem emulovat synchronní komunikaci. Další

nevýhodou je, že použitím MS jako „prostředníka“ vzniká *úzké místo systému*. V případě selhání serveru MS dochází k pádu celého systému. Tento problém je u kvalitních komerčních implementací řešen pomocí klastrování a replikace dat na více serverů. Další nevýhodou může být *nedostatek standardů*, jelikož hlavní standard JMS nedefinuje mnoho důležitých parametrů (formát zpráv, protokol přenosu, atd.) a navíc je víceméně určen pouze pro aplikace vyvíjené v Javě.

Další informace tykající se MOM je možné najít na [3, 4, 5].

2.3 Java Message Service

Informace k této kapitole byly čerpány především z [6, 8, 9].

„Java Message Service (JMS) API je standardem MOM, který umožňuje aplikacím (subsystémům) implementovaných v Javě vytvářet, posílat, přijímat a číst zprávy“ [6]. Definuje společnou sadu rozhraní a sémantiky, které umožňují aplikacím komunikovat s implementacemi messaging systémů [8].

Specifikace JMS byla vytvořena především za účelem usnadnění práce při používání různých implementací MS. Definuje *jednotný způsob* komunikace a práce s jednotlivými implementacemi MS, které podporují tuto specifikaci. V případě nutnosti změny MS by nemělo být nutné provádět změny v kódu aplikace.

Nevýhodou JMS specifikace je, že je k dispozici pouze pro *aplikace napsané v Javě*, *nedefinuje protokol přenosu*, *administrační API*, *zabezpečení*, *a další* [6].

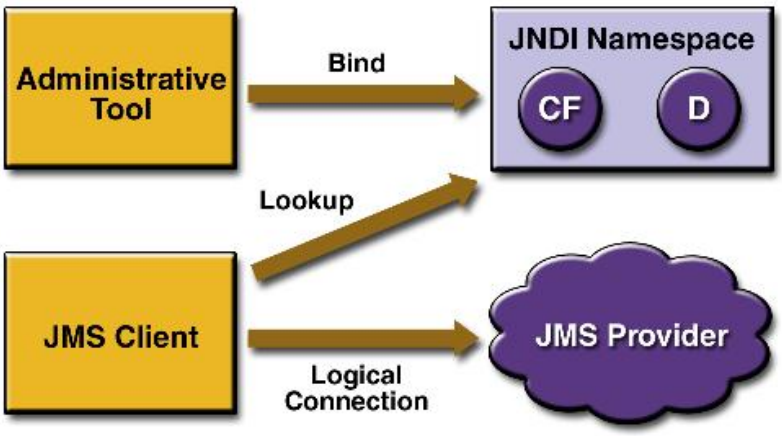
Poslední verze JMS specifikace je 1.1 z roku 2002 a je součástí platformy Java EE.

2.3.1 Architektura JMS API

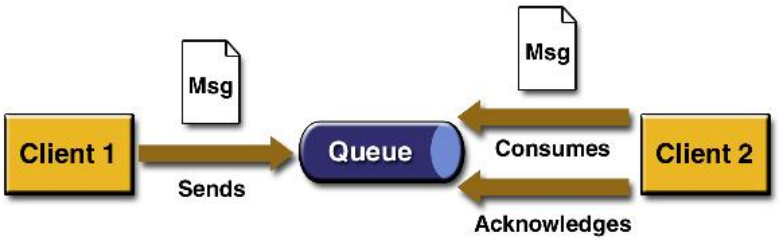
Architektura JMS API se skládá z následujících částí:

- *JMS Provider* – konkrétní implementace MS, která poskytuje JMS rozhraní a další administrační a řídící funkce.
- *JMS Klient (JMS Client)* – aplikace, podsystém či komponenta, která produkuje a konzumuje zprávy a je napsaná v Javě.
- *Administrační objekty (Administered objects)* – JMS objekty, které administrator vytvořil pro použití klienty. Např. Connection Factory – pro vytvoření připojení k JMS Providerovi.
- *Zprávy (Messages)* – objekty, které v sobě přenáší informace mezi klienty.

Na obrázku 2 je zachycena interakce jednotlivých částí. *Administrační nástroje (Administrative tools)* umožňují svázat destinace nebo connection factory s objekty v JNDI jmenném prostoru. JMS klient pak může při vytváření destinací, aj. využívat těchto objektů z tohoto jmenného prostoru.



Obrázek 2: Architektura JMS API; zdroj: [8]



Obrázek 3: Point-to-Point model; zdroj: [8]

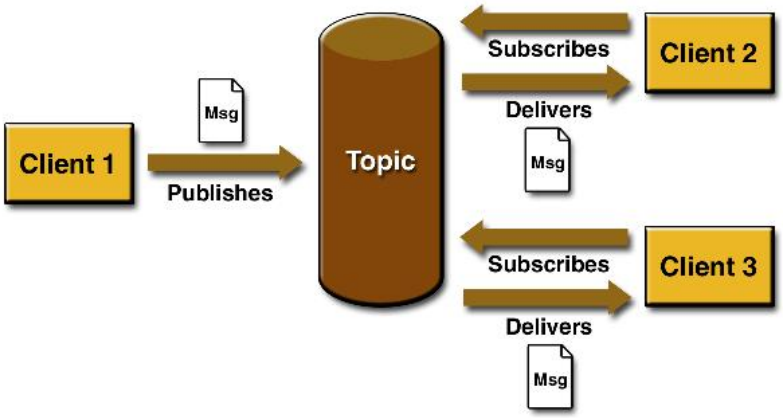
2.3.2 Modely komunikace

V MOM se používají především dva základní modely pro práci se zprávami, respektive pro přenos zpráv. Jsou to *Point-to-Point* a *Publish/Subscribe* model. Oba tyto modely podporuje také JMS specifikace.

2.3.2.1 Point-to-Point model Point-to-Point model je postaven na pojmu *fronta (queue)*. Producenti posílají zprávy do této fronty a konzumenti zpráv si je z této fronty vyzvedávají. Zprávy zůstávají ve frontě, dokud nejsou vyzvednuty nebo dokud nevyexpirují. Zprávy může posílat *více producentů*, ale každá zpráva má *pouze jednoho konzumenta*. Po vyzvednutí zprávy je odstraněna z fronty a žádný další konzument si jí už nevyzvedne. Proto je tento model vhodný pokud chceme, aby zprávy byly zpracovány právě jedním konzumentem.

Point-to-Point model je znázorněn na obrázku 3.

2.3.2.2 Publish/Subscribe model Publish/Subcribe model je postaven na pojmu *topik (topic)*. Producenti posílají zprávy do topiku a *aktuálně připojení konzumenti* okamžitě konzumují tyto zprávy. Na rozdíl od Point-to-Point modelu, zprávy zůstávají v topiku



Obrázek 4: Publish/Subscribe model; zdroj: [8]

jen na dobu nutnou pro distribuci k jednotlivým připojeným konzumentům. Nepřipojení konzumenti v době produkování zprávy zprávu nikdy neobdrží. Výjimku z této „časové závislosti" tvoří *trvalí odběratelé (durable subscriber)*, kteří obdrží zprávu, i když nejsou aktivní při produkování zprávy. V tomto modelu může zprávy konzumovat žáden, jeden i mnoho konzumentů.

Publish/Subscribe model je znázorněný na obrázku 4.

2.3.3 Konzumace zpráv

Messaging je ze své podstaty *asynchronní.* Specifikace JMS používá tento termín v konkrétnějším významu. Zprávy mohou být konzumovány dvěma způsoby:

- *Synchronně* – Konzument si vyzvedne zprávu voláním metody `receive()`. Metoda může čekat, dokud nepřijde zpráva nebo čekat určitý časový interval.
- *Asynchronně* – Klient si registruje *posluchače zpráv (Message Listenery).* Když přijde zpráva do sledované destinace, zavolá se metoda `onMessage(...)` Message Listeneru a dojde ke zpracování zprávy. Zatímco Message Listener čeká na možné zprávy, klient může vykonávat jiné akce [8].

2.3.4 Zprávy JMS

Zprávy JMS se skládají ze tří částí:

- *Hlavička (Message Headers)* – obsahuje hodnoty, které musí nebo můžou být nastaveny, sloužící pro identifikaci a směrování zpráv klienty i JMS Providerem. Např. `JMSDestination`, `JMSMessageID`, atd.
- *Vlastnosti (Message Properties)* – obsahuje další dodatečné informace (např. pro vyfiltrování zpráv) nebo vlastnosti závislé na konkrétní implementaci MS.

- *Tělo (Message Body)* – obsahuje požadovaná přenášená data, které mohou být v mnoha formátech: *TextMessage* (řetězec), *MapMessage* (dvojce jméno – hodnota), *BytesMessage* (proud bajtů), *StreamMessage* (proud primitivních datových typů), *ObjectMessage* (serializovaný objekt), *Message* (prázdná zpráva).

U JMS zpráv je možnost nastavit *mód nakládání se zprávami (Delivery Mode).* Tento mód specifikuje, jak bude se zprávami zacházeno v případě, že dojde k selhání JMS severu.

Jsou definovány dva základní módy:

- *Persistentní (PERSISTENT)* – defaultní, zajišťuje, že v případě selhání serveru nebudou data ztracena. Dochází k ukládání zpráv na disk.
- *Nepersistentní (NON_PERSISTENT)* – nevyžaduje po JMS Providerovi ukládání na disk. Není tím pádem zaručeno, že nedojde ke ztrátě dat.

2.3.5 Potvrzování zpráv a transakce

JMS poskytuje prostředky pro zajištění spolehlivosti doručení zpráv. Dokud není zpráva potvrzena, není považována za doručenou.

2.3.5.1 Potvrzování bez transakcí JMS specifikace poskytuje tyto tři typy potvrzování přijetí zpráv bez transakcí:

- *AUTO_ACKNOWLEDGE* – dojde k automatickému potvrzení každé přijaté zprávy po úspěšném návratu z metody receive() nebo onMessage().
- *CLIENT_ACKNOWLEDGE* – konzument *sám potvrdí* metodou acknowldege() přijaté zprávy. Může tak provést až po úspěšném zpracování zprávy, což je výhoda oproti předchozímu typu.
- *DUPS_OK_ACKNOWLEDGE* – nejméně výkonnostně náročné řešení. Zprávy jsou automaticky potvrzené po určitém množství přijatých zpráv (líné potvrzování). Může tím pádem docházet k příjmu duplicitních zpráv.

2.3.5.2 Transakce *Lokální transakce* jsou další možný způsob potvrzování. Umožňují shlukovat několik operací do jedné atomické jednotky – *transakce*. Když je úspěšná, jsou změny potvrzeny (Commit). Když transakce selže, jsou změny vráceny zpět (Rollback).Toto je vhodné zejména, když je potřeba shlukovat přijaté a odesílané zprávy dohromady. Příkladem může být přijetí zprávy konzumentem a odeslání *potvrzovací zprávy* (confirm message, confirm message je s přijatou zprávou svázána pomocí parametru JMSCorrelationID) odesílateli jako potvrzení, že zpráva byla přijata. V tomto případě je velmi vhodné tyto dvě zprávy spojit do transakce. Lokální transakce není ale možné vytvářet napříč více klienty!

Kromě lokálních transakcí je možné použít *distribuované transakce (XA transakce)*, které poskytují rozsáhlejší prostředky umožňující do transakce zahrnout i práci s jiným distribuovaným systémem, např. databází. Celý proces potvrzování je pak řízen *transakčním manažerem* [10].

2.3.6 Další vlastnosti

Dalšími funkce a vlastnosti poskytované JMS specifikací jsou například:

- *Nastavení doby expirace (Time To Live)*
- *Nastavení priority zpráv*
- *Filtrování zpráv pomocí Message Selectorů*
- *Emulace synchronní komunikace* – např. pomocí třídy QueueRequestor nebo dočasných destinací (Temporary Destinations)
- *Message-Driven Bean (MDB)* – umožňuje asynchronní konzumaci zpráv v Java EE aplikacích.

Více informací je možné se dočíst například v [6] a [10].

3 Využití MOM v systémové integraci

V této kapitole budou popsány možnosti využití messagingu (MOM) v dnešních moderních prostředcích systémové integrace založené především na principu *Service-Oriented Architecture (SOA).* Jelikož tato oblast je velice rozsáhlá a mimo rozsah této diplomové práce, budou pouze velice stručně zmíněny některé základní informace. Více o SOA se lze dozvědět například v [11].

3.1 Integrace s webovými službami

Informace pro tuto kapitolu byly čerpány především z [3].

MOM se ukazuje jako velmi vhodný prostředek pro systémovou integraci, který řeší mnoho problému, které se objevovaly při použití jiných řešení. Nevýhodou ale zůstává *reprezentace dat* (jejich formát, struktura). Jako vhodné se v tomto případě ukázalo propojení s *webovými službami.*

„Webová služba (WS) je platformě a jazykové nezávislý standard pro heterogenní systémovou integraci“ [3]. Zaklad tvoří *SOAP protokol*, který poskytuje prostředek pro výměnu strukturovaných informací mezi uživateli v decentralizovaném prostředí ve formě *XML zpráv*. Pomocí jazyka *WSDL* je pak popsáno rozhraní webové služby [12]. Více o webových službách lze nalézt například na [12].

Webový služby lze využívat několika cestami. Nejčastějším použitím je použití WS na způsob RPC. *Ve spojení s MOM se ale jedná o využití WS jiným způsobem, v souladu s principy SOA*.

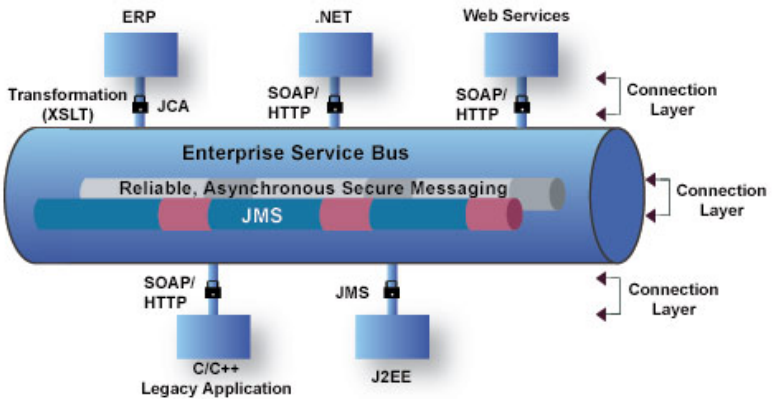
Spojením výhod MOM a výhod webových služeb (nezávislý formát zpráv, jednoduchost transformace dat) odpadá problém messagingu s formáty přenášených dat mezi heterogenními systémy (kde by první systém sice data obdržel od druhého, ale nedokázal by je interpretovat) a vzniká tak flexibilní otevřený systém, který odpovídá principům SOA. Na jednotlivé aplikační procesy se díváme jako na „černé skříňky“ – *služby (services)*. Veškerá komunikace je realizována prostřednictvím *XML zpráv*, které jsou přenášeny mezi jednotlivými službami *asynchronně s pomocí messagingu*. Pokud je formát XML zprávy (která je přenášena mezi dvěma službami) rozdílný, je provedena *transformace* do požadovaného formátu na serveru MS. Snadnější je i vzájemné propojování. Každý systém (např. Java EE aplikace, aplikace v .NETu, zastaralá aplikace v Cobolu, a jiné) je převeden na „XML službu“, která je pak snadno připojitelná i odpojitelná, aniž by muselo dojít ke změnám uvnitř ostatních.

Tato popsaná realizace principů SOA prostřednictvím MOM a WS může tvořit jádro *Enterprise Service Bus (ESB, podniková sběrnice služeb).*

3.2 Enterprise Service Bus

Enterprise Service Bus (ESB) se stala v dnešní době aktuálním trendem v oblasti systémové integrace a vzájemného propojování heterogenních systémů [13].

Dalo by se říci, že ESB je jednou z realizací principů SOA. Klíčovým pojmem je zde *volné propojování služeb.* Pojem ESB není jednoznačně definován. Jednou z definic může



Obrázek 5: Schéma Enterprise Service Bus; zdroj: www.vasanth.in

být: *„Distribuovaná, událostmi řízená architektura orientovaná na služby s důrazem na volnou vazbu, konfigurovatelnost a podporu volných standardů“* [14].

Základním konceptem ESB je tedy *usnadnění připojování aplikací*, *snížení počtu propojení mezi aplikacemi* a *zavedení přídavných hodnot* (např. monitoring) [15].

Základním jádrem ESB je *sběrnice*. Skrze ní si služby vyměňují zprávy. Jedná se o tedy jakýsi komunikační kanál. Hlavním účelem sběrnice je směrování a předávání těchto zpráv. Komunikace na sběrnici probíhá ve standardizované formě. Aby toto bylo možno realizovat, jsou klienti připojování ke sběrnici pomocí *adaptérů*, které poskytují potřebné rozhraní [16].

ESB se skládá ze tří vrstev – *vrstva messagingu*, *vrstva služeb*, *vrstva procesů*, kde vrstva messagingu je nejspodnější vrstva a probíhá zde fyzická manipulace s daty, směrování, atd. Typicky je postavena na *MOM* a *WS* (viz předchozí kapitola) [14, 17].

Schéma ESB a služeb připojených přes adaptéry je možno vidět na obrázku 5.

Mezi typické *vlastnosti* a *výhody ESB* patří:

- *Distribuované řešení* (neexistuje centrální prvek)
- *Standardizované adaptéry a protokoly* (standardizovaná rozhraní pro přístup z jiných technologií či operačních systémů)
- *Transformace dat a kanonický formát zpráv*
- *Messaging* (možnosti synchronní i asynchronní komunikace, komunikační modely)
- *Jednotný management* (monitoring, logování, audity, zabezpečení, atd.)
- *A mnoho dalších* [15].

ESB má i své *nevýhody*. Například částečná závislost na konkrétní implementaci ESB, příliš rozsáhlé řešení pro malé integrace, rychlost zpracování zpráv v XML formátu [14].

Implementací ESB je celá řada, například *Sonic ESB*, *IBM WebSphere ESB*, *Apache ServiceMix*, *Mule*, *a další*.

Další informace o velice rozsáhlém tématu okolo ESB je možné najít například v [18].

4 Existující implementace MOM

4.1 Otevřené implementace

V této kapitole budou zmíněny v dnešní době asi nejznámější otevřené implementace MOM a to *ActiveMQ*, *HornetQ*, *OpenMQ*, *JORAM*, *Apache Qpid*, *RabbitMQ* a *OpenJMS*. Mezi další pak patří například MantaRay, ZeroMQ, SwiftMQ, aj.

4.1.1 ActiveMQ

ActiveMQ je pravděpodobně jednou z nejznámějších a nejoblíbenějších volně dostupných implementací MOM. Společně s HornetQ patří k nejkomplexnějším implementacím, minimálně z těch, které podporují JMS specifikaci a zároveň nejrychleji se rozvíjejícím. Plně podporuje *JMS specifikaci 1.1* a *Java EE 1.4.* Vyvíjený je společností *Apache Software Foundation*, která ho poskytuje jako open source pod licencí Apache Licence 2.0. ActiveMQ je postaven na *STOMP protokolu*, ale má podporu také např. *OpenWire protokolu* [19].

Poslední dostupná stabilní verze v době psaní této diplomové práce byla *ActiveMQ 5.5.1* z dubna roku 2011 [20]. ActiveMQ kromě Javy poskytuje API i pro mnoho dalších programovacích jazyku jako *C*, *C++*, *C#*, *Ruby*, *Perl*, *Python*, *PHP* nebo *Smalltalk* [19]. Je zde také velká podora transportních protokolů, které je možno využít pro přenos zpráv, například TCP, SSL, HTTP, NIO, UDP, JXTA, aj.

Významnou vlastností (aspoň tedy z mého pohledu) je, že destinace jsou *čistě dynamické*. To znamená, že není třeba vytvářet destinace předem v konfiguraci serveru, ale server se o to sám postará v době, kdy je vytvořen požadavek klientem na tuto destinaci. ActiveMQ samozřejmě podporuje *persistentní a nepersistentní přenos zpráv*, možnosti *lokálních* i *XA transakcí.* Dalšími z mnoha poskytovaných vlastností ActiveMQ jsou například *Message Groups* (vytváření skupin exkluzivních odběratelů fronty), *Virtual Topics* (podobné jako Message Groups pro topiky), *Composite Destinations* (přesměrovávání zpráv), *klastrování*, *aj.* [19].

Kromě standardního poskytování podpory JNDI pro vytváření připojení na JMS server a správu destinací, poskytuje ActiveMQ také přímý přístup na JMS server, bez nutnosti využívat JNDI. Pro administraci je poskytováno přehledné webové rozhraní nebo také možnosti využití JMX.

ActiveMQ je také využíváno mnohými projekty, jako například *Apache Camel*, *Apache CXF*, *Apache Geronimo*, *a další*. Také je využíváno ve dvou realizacích ESB – *Apache ServiceMix* a *Mule* [19].

Další možné detaily týkající se ActiveMQ je možné se dočíst na oficiálních stránkách ActiveMQ, viz [19].

4.1.2 HornetQ

Opět velice komplexní implementace messaging systému, obdobně jako ActiveMQ. HornetQ je poměrně nový projekt vyvíjený společností *JBoss*. Poprvé byl uvolněn v srpnu roku 2009 jako nástupce dříve velmi známých a dnes již dále nevyvíjených produktů

JBoss MQ a *JBoss Messaging*. HornetQ je také jako ActiveMQ postavený na Javě a plně podporuje *JMS 1.1.* Poskytován je jako open source pod licencí Apache Licence 2.0 [21]. Podle testů SPECjms2007 provedených v roce 2010 nezávislou výzkumnou skupinou na technické univerzitě Darmstadt HornetQ *překonal o 307% dosavadní výkonnostní rekord.* [22].

Poslední stabilní verze v době psaní diplomové práce byla verze *2.2.5* z června roku 2011. Výhodou HornetQ je, že je standardní součástí *JBoss aplikačního serveru* [21].

Tak jako ActiveMQ je i HornetQ postaven na *STOMP protokolu*. Kromě Javy, pro kterou HornetQ poskytuje jak své vlastní *Core API*, tak *JMS API* (které je v důsledku převáděno na Core API), nabízí HornetQ podporu pro další jazyky. Dále také poskytuje možnost využití různých transportních protokolů, např. TCP, SSL, HTTP, In-VM, atd [21].

Samozřejmostí je zde podpora *persistentních zpráv*, *lokálních transakcí* i *XA transakcí*. HornetQ poskytuje obrovské množství dalších rozšiřujících vlastností, například možnost *hierarchizace destinací*, *možnosti zabezpečení* (integrované s JAAS), *speciální podpora pro posílání rozsáhlých zpráv* (až 8GB zprávy), *Dead letter addresses* (místo pro odkládání zpráv, které se nepodařilo doručit), *Message buffering* (pro zvýšení výkonu), *přesměrování zpráv*, *možnosti klastrování*, *atd.* [21]. Pro správu serveru je možno využít několika možností, které jsou blíže popsány v dokumentaci (JMX managment, atd.) nebo využít také webovou administrační konzoli JBoss aplikačního serveru.

HornetQ je využíván v produktech společnosti JBoss. Příkladem je již zmíněná integrace v *JBoss Application Serveru* nebo jako součást *JBoss ESB*.

Další možné detaily o HornetQ je možné se dočíst na stránkách HornetQ, viz [21].

4.1.3 OpenMQ

OpenMQ je multiplatformní produkt vyvíjený společností *Sun Microsystems*, který je napsán v Javě a plně podporuje *JMS specifikaci 1.1* a také *Java EE 1.4.* OpenMQ je volně dostupný pod licencí CDDL (Common Development and Distribution License) a GPL (GNU General Public License) [23].

Poslední dostupná verze je *OpenMQ 4.5* z března roku 2011. Kromě JMS API poskytuje OpenMQ i API pro práci v jazyce *C/C++* [23].

Obdobně jako ActiveMQ není nutné vytvářet destinace předem v konfiguračním souboru, ale jsou vytvořeny *dynamicky* při jejich prvním použití. Standardně poskytuje podporu po *persistencí zprávy*, *lokální transakce* i *XA transakce.* Dále poskytuje další rozšiřující funkce, jako například: *uchovávání nedoručených zpráv*, *komprimace těla zpráv*, *možnosti zabezpečení přenosu*, *možnosti klastrování*, *aj.* Na rozdíl od ActiveMQ a HornetQ neposkytuje žádné speciální prostředky pro přenos velkých zpráv. Pro správu serveru poskytuje OpenMQ přehlednou GUI aplikaci. Pro rozsáhlejší zprávu je pak možné použít JMX API [23].

OpenMQ je základní JMS provider v aplikačním serveru *GlassFish* a zároveň může být využíván jako součást dvou ESB: *Mule* a *Open ESB* [23].

Další možné detaily týkající se OpenMQ je možné se dočíst na [23].

4.1.4 JORAM

Java Open Reliable Asynchronous Messaging (JORAM) je na Javě postavený messaging systém, který plně podporuje *JMS specifikaci 1.1.* Je vyvíjený od roku 1999 společností *ScalAgent D.T.* pod záštitou konsorcia *OW2*. Volně dostupný je pod licencí LGPL. Na rozdíl od předchozích produktů, využívá JORAM protokol *AMQP* [24].

V době psaní diplomové práce byl poslední dostupná verze *5.7.0* ze září roku 2011. Kromě podpory JMS API pro Javu, poskytuje také API pro *C/C++ klienty* (Xoram API) [24].

JORAM na rozdíl od zbylých MS, které zde jsou zmíněny, *v defaultním nastavení neposkytuje persistentní způsob uchovávání zpráv.* To je nutno nastavit explicitně v konfiguračních souborech. Také neposkytuje jinou možnost připojení k serveru a práce s destinacemi než s využitím JNDI. Standardem ovšem je poskytování jak *lokálních*, tak *XA transakcí*. Dalšími poskytovanými rozšířeními jsou například: *zabezpečení (s využitím JAAS)*, *klastrování*, *podpora replikace dat*, *atd.* JORAM neposkytuje možnost komprimace těla zpráv a možnost dynamického vytváření destinací [24].

Komunikace mezi serverem a klientem může být zajišťována především pomocí In-VM, TCP, SSL. Správa serveru a destinací byla původně poskytována pouze prostřednictvím JMX, od října roku 2011 JORAM nově začal poskytovat navíc možnost správy pomocí webové konzole [24].

JORAM je defaultním JMS providerem v *JOnAS aplikačním serveru*, open source projektu *FraSCAti* a také v implementaci ESB – *PEtALS* od OW2 [25].

Další možné detaily lze nalézt na oficiálních stránkách tohoto produktu, viz [24].

4.1.5 Apache Qpid

Apache Qpid je po ActiveMQ druhou volně dostupnou implementací MOM od společnosti *Apache Software Foundation.* Je poskytován pod licencí Apache Licence 2.0. Na rozdíl od ActiveMQ je postaven na protokolu *AMQP*. Kromě serveru (Message brokeru) napsaného v Javě, poskytuje i *server napsaný v C++*. Navíc umožňuje přístup klientům napsaných v *Javě*, *C++*, *Ruby*, *Pythonu*, *.NETu*. Pro klienty napsané v Javě poskytuje standardní přístup pomocí *JMS API* [26].

Apache Qpid je poměrně nový, ale rychle se rozvíjející produkt. Poslední stabilní dostupná verze v době psaní této diplomové práce byla *Apache Qpid 0.14* z ledna roku 2012.

Apache Qpid podporuje standardní vlastnosti, jako je *persistence zpráv*, *lokální* i *XA transakce*. Navíc poskytuje také možnost odesílání jedené zprávy do více front (*replikaci zpráv*), *možnosti zabezpečení* (autentizace, autorizace, šifrování), *klastrování* a mnoho dalších vlastností, viz [26].

Správa serveru napsaného v Javě je možná prostřednictvím JMX nebo pomocí naistalovaného plug-inu do vývojového prostředí Ecplise. Apache Qpid je možné integrovat s ESB *Mule* [27].

Více informací lze nalézt na [26].

4.1.6 RabbitMQ

RabbitMQ je robustní, multiplatformní implementace MOM vyvíjená v programovacím jazyku *Erlang* společností *VMware*. Volně dostupný je pod licencí Mozilla Public License. Je postavený na protokolu *AMQP* [28].

RabbitMQ je vyvíjený od roku 2007 a poslední dostupná stabilní verze v době psaní této diplomové práce byla *2.7.1* z prosince roku 2011. RabbitMQ má podporu pro obrovské množství programovacích jazyků jako *Java*, *Ruby*, *Python*, *PHP*, *.NET*, *Perl*, *C/C++*, *Erlang*, *Haskell*, *atd.* Pro Javu poskytuje především své *vlastní API*. Také je zde určitá možnost využití *JMS API*. Samozřejmostí je u RabbitMQ podpora *lokálních* a *distribuovaných transakcí*, *persistence dat*, *možnosti zabezpečení*, *klastrování*, *aj.* [28].

Pro správu poskytuje RabbitMQ webovou konzoli nebo případně správu přes příkazovou řádku.

RabbitMQ může být využíván jako součást ESB: např. *Mule*, *Apache Synapse* [28].

Podrobnější informace o RabbitMQ je možno nalézt na [28].

4.1.7 OpenJMS

OpenJMS je poměrně známá a již dnes nevyvíjená implementace messaging systému postavená na Javě s podporou *JMS 1.1.* Vyvíjen byl společností *Sun Microsystems* od roku 2000 do roku 2006. Poslední dostupná verze je *beta verze 0.7.7* [29].

Tento MS podporuje pouze málo vlastností. Umožňuje *persistentní* a *nepersistentní přenos zpráv*, ale jako persistentní mohou být přenášeny zprávy pouze do velikosti 32KB. Poskytuje podporu *lokálních transakcí*, ale ne XA transakcí. Neposkytuje žádné vlastnosti „vyšší úrovně“, jako například podpora přímého přístupu k serveru bez JNDI, klastrování, JMX management, kompresi těla zprávy, API pro jiné jazyky než Java. Také je podporována pouze autentizace, ale ne autorizace uživatelů [29].

Pro komunikaci s klientem OpenJMS podporuje tyto protokoly: TCP, RMI, HTTP a SSL.

Více informací o produktu OpenJMS lze nalézt na [29].

4.2 Komerční implementace

V této kapitole budou stručně zmíněny některé nejznámější komerční implementace MOM. Jsou to *FioranoMQ*, *SonicMQ* a *WebSphere MQ* a jejich použití. K dalším komerčním implementacím patří například Tibco EMS, Oracle AQ, MSMQ, aj.

4.2.1 FioranoMQ

FioranoMQ je komerčním produktem, který je vyvíjený společností *Fiorano*. Podle společnosti Fiorano se jedná o *dlouhodobě nejvýkonnější messaging systém v dnešní době*, což dokládá sice starší, ale nezávislá studie zabývající se výkonností JMS serverů provedená na univerzitě ve Würzburgu z roku 2006 [32]. Toto také potvrzují porovnání, které skoro každoročně provádí sama společnost Fiorano. Aktuální z roku 2011 lze nalézt zde [30].

FioranoMQ je implementováno v Javě s podporou *specifikace JMS 1.1* a *Java EE 1.4.* Poslední stabilní verze v dnešní době je *FioranoMQ 9.4.1* z prosince roku 2011. Kromě podpory pro klienty napsané v *Javě* poskytuje FioranoMQ také podporu pro *C/C++*, *C#*, *Visual Basic*. A také použití různých protokolů: HTTP, HTTPS, SSL, SOAP, atd. [32].

FioranoMQ poskytuje všechny „standardní vlastnosti" požadované od dnešních implementací MOM, jako je *persistence dat*, *lokální* a *distribuované (XA) transakce*, *komprese těla přenášených zpráv*, *klastrování*, *aj.* Za zmínku stojí velice rozsáhlé *možnosti zabezpečení* (autentizace, autorizace s podporou JAAS a XACML, plug-in pro spolupráci s různými externími systémy (např. LDAP), podpora pro digitální certifikáty, atd.) [32].

Pro správu serveru a destinací je k dispozici API založené na JMX a možnost použití vizuálních nástrojů . Dále jsou poskytované vlastnosti pro správu logů, událostí, diagnostiku problémů, atd. Dokumentace, která je k FioranoMQ poskytována je velice rozsáhlá, což také vypovídá o komplexnosti tohoto řešení.

FioranoMQ je především součástí dalších projektů společnosti Fiorano, jako například *Fiorano ESB* nebo *Fiorano SOA Platform*. Také je využito v mnoha významných systémech velkých firem, např. *Fairex*, *FinScope*, *IS-Australia for Toyota*, *University of California*, *aj.* [32].

Více informací o FioranoMQ je možné nalézt na [32].

4.2.2 SonicMQ

SonicMQ je další populární komerční implementací messaging systémů. Je vyvíjen a poskytován společností *Progress Software*, která se zabývá oblastí systémové integrace. Jedná se o velice robustní a komplexní řešení využívané v dalších produktech od Progress Software. Podle výkonnostních testů prováděných společností Fiorano je SonicMQ *druhým nejvýkonnějším produktem* hned za FioranoMQ [30].

SonicMQ je naimplementován v Javě s podporou *specifikace JMS 1.1.* Kromě podpory pro *Java klienty* nabízí také API pro klienty napsané v *C/C++*, *C#* a *Visual Basicu*. Podporovanými protokoly pro komunikaci mezi klientem a serverem jsou HTTP, HTTPS, TCP, SSL [33]. Poslední nabízená verze je *SonicMQ 8.5* z roku 2011.

Kromě dnes standardních podporovaných vlastností (persistence dat, lokální a distribuované (XA) transakce, klastrování, aj.) poskytuje SonicMQ *detekci duplicitních zpráv*, *dynamického směrování zpráv* (Dynamic Routing Architecture) spolu s možností *automatizované centrální správy distribuovaných konfigurací* [34], *integraci s již existující bezpečnostní infrastrukturou*, *nepřetržitou dostupnost* zajištěnou replikací dat mezi primárními a sekundárními message brokery, rozsáhlé možností správy serverů a destinací, atd. O dalších vlastnostech a funkcích je možné se dočíst na [33].

SonicMQ je součástí dalších produktů společnosti Progress Software (*Sonic ESB*, *aj.*). SonicMQ je samozřejmě také nasazen v systémech některých známých společností, např. *Vodafone*, *CERN* [33].

4.2.3 WebSphere MQ

WebSphere MQ je známou a již poměrně dlouho se rozvíjející komerční implementací MOM. První verze byla uvolněna v roce 1992 pod názvem *MQSeries*. WebSphere MQ je jedním z produktů rodiny *WebSphere* od firmy *IBM* [35].

V době psaní této diplomové práce byla poslední dostupná verze *WebSphere MQ 7.1* z listopadu roku 2011. WebSphere MQ poskytuje API pro přístup klientům napsaných v řadě různých programovacích jazyků (*Java*, *.NET*, *C/C++*, *COBOL*, *Perl*, *Python*, *Windows PowerShell*, *aj.*). Pro Javu je kromě nativního API samozřejmě poskytována podpora *JMS API* [35, 36].

WebSphere MQ je velice robustní a komplexní řešení s rozsáhlými možnostmi provázání s dalšími produkty od IBM. I zde je samozřejmostí podpora „standardních vlastností“ požadovaných od dnešních implementací MOM (viz předchozí produkt), *podpora pro integraci s webovými službami*, *zamezení příjmu duplicitních zpráv*, *atd.* Kromě standardních prostředků zabezpečení je poskytováno využití *certifikátů* nebo *elektronického podpisu* [35].

WebSphere MQ je úspěšně využíváno v obrovské řadě systémů firem a institucí, například *City Bank* [37], *National Geographic Society*, *New Zealand Transport Agency*, *DONG Energy* [35].

Další možné informace týkající se IBM WebSphere MQ je možné se dočíst na [35].

5 Realizace testovací aplikace

Cílem praktické části mé diplomové práce byla realizace testovací aplikace, na které by bylo možno provádět výkonnostní testy jednotlivých vybraných otevřených implementací messaging systémů. Cílem bylo tuto aplikaci navrhnout tak, aby splňovala požadavky na použití MOM (messaging systémů), tj. aby byla založena na vzájemné výměně dat (zpráv), která pak bude realizována právě pomocí MOM.

Testovací aplikace měla být navržena a implementována tak, aby realizovala nějakou smysluplnou „reálnou“ činnost. Nemělo tedy jít o pouhou velmi primitivní aplikaci, na které by se samozřejmě také daly provádět výkonnostní testy, ale o program, který již využívá MOM v širším a smysluplnějším kontextu.

V úvahu přicházely dvě možná řešení. Buď to více aplikací, kde každá realizuje jinou činnost (např. jedna vytváří nějaké elektronické dokumenty, druhá tyto dokumenty převádí do pdf formátu, atd.), kde vlastně tyto aplikace jsou ve vzájemné interakci a to právě skrze messaging systém. Druhou možností bylo vytvoření jedné aplikace, která by pak v reálném nasazení byla rozmístěna jako klient na větším počtu stanic a vzájemně by tito klienti spolu komunikovali srze messaging systém. Nakonec byla vybrána tato druhá možnost.

5.1 Motivace

Jak bylo zmíněno výše, vyvíjená testovací aplikace bude pouze jedna s tím, že se bude jednat o jakýsi typ tlustého klienta, který by v reálu byl rozmístěn na velkém počtu stanic s tím, že jejich vzájemná komunikace a přenos dat mezi nimi *se bude realizovat prostřednictvím MOM.*

Při vymýšlení vhodné a „smysluplné“ aplikace byla nalezena motivace v kombinaci *aplikací pro sociální sítě* (konkrétně Twitter), *aplikací typu Instant Messagingu* (konkrétně třeba Jabberu, který může sloužit i pro vzájemnou komunikaci programů mezi sebou) a *emailové komunikace.*

Všechny tyto přístupy jsou založeny na principu existence množiny uživatelů, kteří jsou spolu navzájem nějak provázáni a interagují spolu. Uživatelé si mohou posílat zprávy – posílat zprávu jen jednomu svému příteli nebo třeba hromadnou zprávu. Číst si to co jiní uživatelé zveřejní a nebo také posílat žádosti na nové přátelství, atd.

Má aplikaci je také založena na tomto principu. Jedná se zde o tlustého klienta, ke kterému se přihlašují uživatelé. Tito uživatelé pak mají své přátele, kterým můžou posílat zprávy a to jak *textové*, tak nějaké větší *elektronické dokumenty.* Tyto zprávy mohou posílat jak jednomu svému příteli, tak více přátelům. Na rozdíl od zmíněných motivačních přístupů zde nebude žáden centrální server ve smyslu aplikačního serveru, který by zprostředkovával komunikaci nebo databáze která by uchovávala zprávy (odeslané, přijaté, přečtené, . . .). Veškerý přenos těchto zpráv mezi uživateli (přihlášených na konkrétní stanici) bude realizován prostřednictvím *messaging systému*. Více o struktuře a principu aplikace v další kapitole 5.2.

Na vyvíjenou aplikaci je možné se také dívat jako Twitter s dokumenty. Bude poskytovat podobné funkce jako Twitter s tím, že bude přenášet nejen textové zprávy, ale i soubory o velikosti několika stovek KB.

Motivující zajímavostí také je, že sám Twitter ve svém backendu využívá message queue a to například pro přenos zpráv mezi jeho distribuovanými servery. Konkrétně si tuto komunikaci (message queue) píšou vlastními silami, viz [38].

5.2 Princip aplikace

V této kapitole je popsán základní princip testovací aplikace bez ohledu na implementační prostředí a programovací jazyk, volbu databáze a podobně.

Jak už bylo mnohokrát zmíněno výše, testovací aplikace bude aplikace na způsob *tlustého klienta*. V reálu by se prostřednictví tohoto klienta uživatelé přihlašovali do systému. Přihlášený uživatel pak může provádět činnosti odesílání *textových* nebo *větších elektronických dokumentů jiným uživatelům*, se kterými je ve vztahu. Posílání těchto zpráv bude, na rozdíl od výše zmíněných motivačních přístupů, *realizováno prostřednictvím MOM*. Princip asynchronních messaging systémů je takový, že doručené zprávy jsou ze serveru smazány[1]. Z toho vyplývá, že zprávy, které byly uživatelem přijaty, jsou již ztraceny a neuchovávají se[2].

Pro fungování aplikace je nutno si uchovávat informace o uživatelích, především to jestli již nejsou na jiné stanici přihlášeni a pak samozřejmě informace o vzájemných vazbách. Tyto informace jsou uchovávány v databázi.

Jak je jistě také patrné, aplikace samozřejmě *nebude umožňovat přihlašování skutečných uživatelů*, a tím pádem ani výběr činností, které by normální uživatel podle svého uvážení po přihlášení prováděl (jako například: přečtení si přijatých, odeslání textové zprávy o nějaké délce nějakému svému příteli, atd.). *Vše bude samozřejmě simulováno s využitím náhodných generátorů pro jednotlivé činnosti a to přímo v aplikaci.*

Samotný princip běhu aplikace je několikanásobné spuštění klientské aplikace. Na každé aplikaci dojede k náhodnému vygenerování uživatele, jeho přihlášení (pokud ještě není přihlášen na jiném klientovi), spuštění přijímání příchozích zpráv, cyklické volby typu zprávy k odeslání a její odeslání na server MS (JMS server). Po určitém časovém intervalu dojde k odhlášení uživatele a přihlášení jiného a opět následuje totéž co u předchozího.

5.3 Realizované funkce

V této kapitole si popíšeme funkce či činnosti, která aplikace realizuje, jejich vztah k JMS specifikaci a jejich analogii k funkcím poskytovaných Instant messagingem nebo aplikacemi typu Twitter. Realizovanými funkcemi jsem chtěl využít a v konečném důsledku také otestovat všechny hlavní možnosti, které umožňuje JMS specifikace a které implementují i všechny testované messaging systémy.

[1] Toto platí v základním nastavení MS, některé implementace MS umožňují např. replikaci dat

[2] V reálu, pokud by měla být skutečně nasazena, by to byl samozřejmě problém, zde v testovací aplikaci je to spíše výhoda

Realizované funkce:

1. *Posílání textové zprávy jednomu uživateli*
   - V analogii k Twitteru nebo Facebooku se jedná o poslání přímé zprávy jednomu konkrétnímu uživateli. Stejně tak u Instant Messagingu (IM).
   - Vzhledem k JMS specifikaci se jedná o poslání TextMessage do fronty (queue).
2. *Posílání elektronického dokumentu jednomu uživateli*
   - Twitter zatím neumožňuje posílání dokumentů. Z hlediska IM je částečnou analogií přímý přenos souborů, kde ale ve většině případů musí být oba uživatelé přihlášeni.
   - Vzhledem k JMS specifikaci se jedná o poslání BlobMessage[3] do fronty.
3. *Posílání textové zprávy aktuálně přihlášeným uživatelům*
   - Poslání zprávy více nebo všem uživatelům, kteří jsou aktuálně přihlášeni.
   - V analogii k IM by se mohlo jednat částečně o hromadný chat více uživatelů.
   - Vzhledem k JMS specifikaci se jedná o ukázku posílání TextMessage do topiku (topic).
4. *Posílání elektronického dokumentu přihlášeným uživatelům*
   - Poslání zprávy s dokumentem více nebo všem uživatelům, kteří jsou zrovna přihlášeni.
   - Vzhledem k JMS specifikaci se jedná o poslání BlobMessage do topiku.
5. *Posílání textové zprávy více uživatelům*
   - Poslání textové zprávy více nebo všem uživatelům, kteří nemusí být přihlášeni.
   - Vzhledem k IM nebo emailu se může jednat o rozeslání hromadné zprávy více uživatelům. Vzhledem k Twitteru nebo Facebooku se může jednat o publikování zpráv na „zdi“.
   - Vzhledem k JMS se jde o asynchronní využití topiku, na kterém jsou připojeni trvalí odběratelé (durable subscriber). Na tento topik je odeslána TextMessage.
6. *Posílání elektronického dokumentu více uživatelům*
   - Poslání zprávy s dokumentem více nebo všem uživatelům, kteří nemusí být přihlášeni.
   - U IM nebo emailu obdobné jako předchozího. U Twitteru, Facebooku nebo jiných sociálních sítí je analogií sdílení a publikování dokumnetů, fotek.

[3] Pro různé implementace MS jsou pro toto různé pojmenování i způsoby realizace (např. StreamMessage).

- Vzhledem k JMS se jedná opět o asynchronní využití topiku, na kterém jsou připojeni trvalí odběratelé. Na tento topik je odeslána BlobMessage.

7. *Posílání textové zprávy nebo elektronickému dokumentu přihlášeným uživatelům*
   - Jedná se o obdobu 3. a 4. bodu, s tím rozdílem, že zpráva je poslána všem přihlášeným uživatelům bez rozdílu toho, zda existuje nebo neexistuje vazba mezi odesílatelem a příjemcem.
   - Částečná analogie v jiných systémech může být zpráva, která má informovat o něčem všechny uživatele bez rozdílu.
   - Tyto dvě činnosti byly přidány především z důvodů jejich pozdějšího využití při testování výkonnosti určitých vlastností (škálovatelosti).
   - Z hlediska JMS specifikace se jedná opět o využití topiku.
8. *Posílání žádosti o přátelství*
   - Analogie u IM nebo aplikací pro sociální sítě je zřejmá.
   - Z hlediska JMS specifikace je to především MapMessage posílaná do fronty.
9. *Posílání potvrzovacích zpráv*
   - Jedná se o odesílání potvrzovací zprávy příjemcem odesílateli, čímž odesílatel získá jistotu, že zpráva byla úspěšně přijata.
   - Vzhledem k JMS specifikaci se jedná o vyzkoušení funkčnosti parametru replyDestination, který uchovává informaci o destinaci, na kterou se mají posílat potvrzovací nebo i jiné zprávy (například v mé aplikaci je toto využito k posílání odpovědi na zprávu žádosti o přátelství).
10. *Přijímání zpráv*
    - Z hlediska JMS je to zaměření na příjem různých typů zpráv (TextMessage, BlobMessage, MapMessage, Message) a využití MessageListenerů.

5.4 Návrh front a topiků

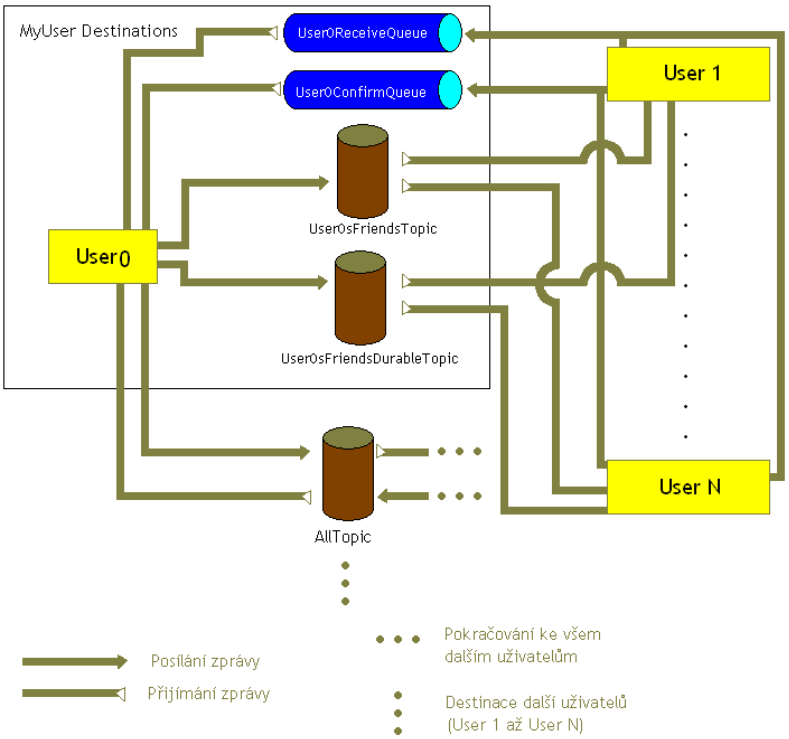
V předchozích dvou kapitolách byl popsán princip a funkce testovací aplikace. V této kapitole budou popsány fronty a topiky, které je nutno na JMS serveru vytvořit.

Každému uživateli aplikace přísluší pevně daný počet destinací (front a topiků) na JMS serveru. Tyto destinace jsou vytvořeny při jeho vytvoření v systému nebo při prvním použití této destinace[4].

Pro každého uživatele existují tyto destinace:

1. *UserNameReceiveQueue* – jedná se o frontu, kde uživatel *přijímá zprávy*, které mu *poslali* do této fronty *ostatní uživatelé*, se kterými je ve vztahu (přátelé). Pomocí této fronty jsou realizovány výše zmíněné funkce 1, 2, 8 a 10.

[4]Toto záleží na konkrétní implementaci messaging systému



Obrázek 6: Struktura destinací testovací aplikace

2. *UserNameConfirmQueue* – jedná se frontu, kde uživatel *přijímá potvrzovací zprávy*, které mu *poslali ostatní uživatelé*, kterým předtím poslal nějakou zprávu. Pomocí této fronty jsou realizovány funkce 9 a 10.

3. *UserNamesFriendsTopic* – jedná se o topik, kam uživatel *odesílá své zprávy*, které pak mohou *přijímat jeho přátelé*, kterým je zpráva určena. Pomocí tohoto topiku jsou realizovány funkce 3, 4 a 10.

4. *UserNamesFriendsDurableTopic* – jedná se o obdobu předchozího s tím, že se může jednat o *„asynchronní komunikaci“*. Uživatelé, pro které je zpráva určena, nemusí být zrovna přihlášeni. Podmínkou je, že uživatel je na tomto topiku připojen jako *trvalý odběratel (durable subscriber)*. Pomocí tohoto topiku jsou realizovány funkce 5, 6 a 10. Tyto čtyři destinace existují pro každého vytvořeného uživatele. Kromě nich existuje ještě:

5. *AllTopic* – který je pouze jeden pro celý systém. Do tohoto topiku mohou *posílat* své zprávy *všichni uživatelé* a *všichni uživatelé je mohou také přijímat*, což odpovídá funkci číslo 7.

Poznámka: Za *UserName* přijde samozřejmě jméno konkrétního uživatele.

Celá struktura provázání destinací a uživatelů je znázorněna na obrázku 6. V obrázku by bylo velice obtížné znázornit všechny provázání z důvodů následné velké nepřehlednosti. Proto jsem znázornil tuto strukturu především z hlediska jednoho uživatele (User 0). Z hlediska ostatních uživatelů jsem znázornil pouze jejich vztah k destinacím nultého uživatele a AllTopicu.

Ohraničení *MyUser Destinations* ohraničuje tedy destinace svázané s existencí konkrétního uživatele. Jsou to *User0ReceiveQueue*, *User0ConfirmQueue*, *User0sFriendsTopic* a *User0sFriendsDurableTopic*.

Na schématu je vidět, že User0 z User0ReceiveQueue, User0ConfirmQueue zprávy *přijímá*. Ostatní uživatelé do nich zprávy *posílají*.

Do User0sFriendsTopic a User0sFriendsDurableTopic naopak zprávy *posílá sám User0* a *všichni ostatní uživatelé* (pokud teda jsou spolu ve vztahu) z těchto topiků zprávy *přijímají*. Speciálním případem je AllTopic do kterého posílají a z kterého přijímají všichni uživatelé.

Každý uživatel tedy naslouchá jednak na svých ReceiveQueue, ConfirmQueue a na AllTopicu. A pak *na všech* FriendsTopic a FriendsDurableTopic svých přátel.

5.5 Popis aplikace z hlediska JMS specifikace

V této kapitole bude popsáno, jak jednotlivé základní prvky JMS specifikace jsou promítnuty do vyvíjené aplikace. Tímto je myšleno například persistence a podobně. Nebudou zde rozebírány věci, které jsou „speciální“ a jsou rozličné pro každou implementaci messaging systémů.

5.5.1 Posílání a přijímání zpráv

Připojení na sever (Connection) je v aplikaci využíváno pro vytváření odesílatelů zpráv (*MessageProducerů*) a následného odesílání zpráv na server a příjemců (*MessageConsumerů*) a následného přijímání zpráv ze serveru.U některých nasazených implementací také pro provádění administračních operací.

U odesílání zpráv jsem se rozhodl pro vytváření nového připojení a nového MessageProducera *pro každou novou odesílanou zprávu*. Aplikace by zbytečně držela připojení a přitom by uživatel třeba dlouho nechtěl nic odesílat. U přijímání zpráv je pak připojení vytvořeno *při zavolání metody na začátek přijímání zpráv* a ukončeno *při zavolání metody na ukončení přijímaní zpráv ze serveru.* Na základě připojení je pak nutno vytvořit pro každou destinaci, z které má být přijímáno, nového MessageConsumera.

Jak již bylo podrobně rozebráno v kapitole 2.3 existují dva druhy přijímání zpráv: *synchronní* (volání metody receive()) nebo *asynchronní*, kde je nejlepší variantou použití MessageListeneru. Ve své testovací aplikaci jsem použil na veškerý příjem zpráv ze všech možných destinací právě asynchronní přístup pomocí MessageListenerů. Každý vytvořený MessageConsumer může mít k sobě připojeného právě jednoho MessageListenera. Použití synchronního přístupu by sice bylo možné, ale pro účely pozdějšího testování *ne příliš vhodné*.

5.5.2 Persistence

Parametr DeliveryMode udává, jestli zpráva přežije pád či vypnutí serveru. V aplikaci jsem nastavil pro veškeré posílané zprávy *persistentní mód*, kromě zpráv směřujících do netrvalých topiků, kde tento parametr z jejich principu nehraje žádnou roli.

5.5.3 Potvrzování zpráv, transakce

Potvrzovacích režimů po odeslání nebo přijmu zprávy je celá řada (viz kapitola 2.3). V aplikaci je použito *transakční potvrzování*. Dále byly použity *potvrzovací zprávy (confirm message)*, které slouží k informování odesílatele zprávy, že příjemce jí úspěšně přijal. V aplikaci jsou confirm message použity pro potvrzení všech zpráv, které byly odeslány do *fronty nebo trvalého topiku s připojenými konzumenty.*

5.5.4 Priorita zpráv

Čím vyšší priorita (parametr Priority), tím přednostněji bude zpráva doručena. *Pro zprávy posílané do topiků* byla nastavena *nejvyšší priorita* rovna 9, jelikož je nutno tyto zprávy přijímat dokud jsou vysílány a není vhodné v tuto dobu upřednostňovat jiné zprávy, které je možno přijímat i později. *Pro zprávy požadavku na přátelství* byla nastavena priorita 8 a *odpovědi na tento požadavek* priorita 7. *Pro zbylé zprávy posílané do fronty nebo trvalého topiku s připojenými konzumenty* byla nastavena náhodně priorita 4 nebo 6 (nižší a vyšší priorita zprávy). *Pro obyčejné potvrzovací zprávy* pak priorita 3.

5.5.5 Filtrování zpráv

Pro filtrování zpráv se používají Message selektory, které byly popsány v kapitole 2.3. Message selektory byly v aplikaci využity k výběru uživatelů, kterým je zpráva určena. Do proměnné nazvané ForUser pro konkrétní zprávu byly vloženy jména uživatelů, pro které je zpráva určena. V případě, že zpráva je pro všechny uživatele, bylo vloženo slovo „ALL“. Na straně přijmu pak byly pomocí Message selektoru filtrovány zprávy, které jsou určeny danému uživateli.

5.6 Návrh a implementace aplikace

V předchozích několika kapitolách byla vyvíjená testovací aplikace popisována z hlediska jejích funkcí a struktury a to především z pohledu MOM a JMS (návrh destinací, propojení s uživateli, . . .). V této kapitole bude popsána z hlediska implementačního.

5.6.1 Vývojové prostředí a technologie

Testovací aplikace je *konzolová aplikace* implementovaná v jazyce *Java*. Pro uchovávání potřebných dat tj. informace o uživatelích a jejich vzájemných vazbách jsem použil *MySQL 4.1 databázi*. Pro přístup k databázi jsem využil *JDBC API* (konkrétně byl použit driver mysql-connector-java5.1.6-bin.jar, který je součástí vývojového prostředí NetBeans 6.9.1).

Celá aplikace byla vyvíjena ve vývojovém prostředí *NetBeans 6.9.1*. Dále byly využívány *API jednotlivých nasazovaných implementací messaging systémů*. Všechny nasazované messaging systémy podporovaly poslední verzi *JMS API 1.1* vydanou v roce 2002. Nutno zde ale podotknout, že každý tento JMS Provider[5] si tuto specifikaci rozšířil o další vlastnosti a funkce. To se týká především velice různého realizování připojení k jednotlivým JMS serverům, či vztahu k vytváření destinací. Z tohoto důvodu jsem také navrhl a implementoval adaptér (*JMS Adaptér*), který pak testovací aplikaci umožňuje skoro jednotným způsobem přistupovat k jednotlivým JMS Providerům a pracovat s nimi (viz kapitola 6). I přesto bylo vhodné (hlavně vzhledem k pozdějšímu testování) vytvořit *různé verze aplikace pro různé JMS Providery*, i když jen s nepatrnými změnami.

5.6.2 Návrh databáze

Databáze s názvem *Diplomka* obsahuje pouze dvě tabulky:

- *User (id_user, name, login)* – uchovává informace o uživatelích, kde *name* je uživatelské jméno uživatele a *login* uchovává informaci o tom, zda je uživatel zrovna přihlášen nebo nikoli a nabývá hodnoty 0 nebo 1.
- *Friends (id_user, id_friend)* – uchovává informace o vazbách mezi uživateli, kde *id_user* i *id_friend* jsou cizí klíče z tabulky *User*.

5.6.3 Návrh aplikace

Jak už bylo vícekrát zmíněno výše, očekává se, že v jeden okamžik poběží více instancí téže aplikace na jedné nebo více klientských stanicích. Na každé z těchto stanic je přihlášený nějaký uživatel, který skrze tuto aplikaci posílá své zprávy na JMS server nebo je ze serveru přijímá. Komunikace aplikace a serveru neprobíhá přímo, ale *přes naimplementovaný JMS Adaptér* (viz další kapitola). Toto zachycuje obrázek 7. Statická struktura aplikace je zachycena pomocí třídního diagramu na obrázku 8.

Nyní zde budou popsány jednotlivé třídy aplikace, jejich činnost a význam.

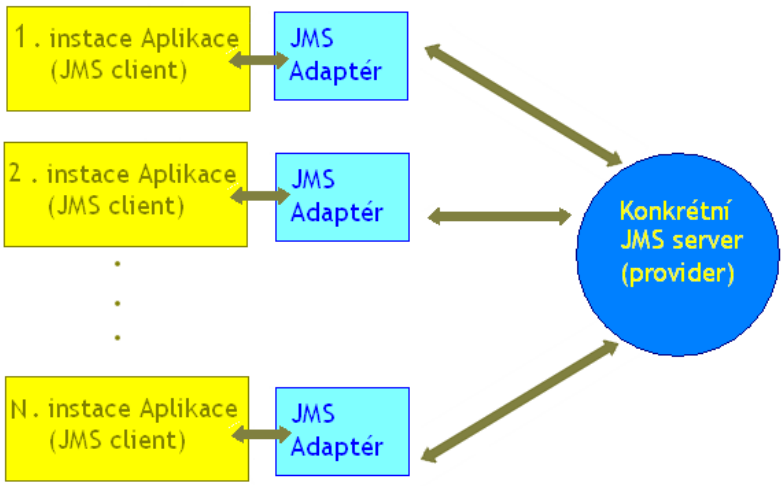
5.6.4 Třída Main

Je „startovní třída“ aplikace. Probíhá zde nastavování JMS Providera (informace pro JMS Adaptér), nastavování parametrů aplikace předané přes příkazovou řádku a vytváření instancí vláken třídy Aplication a jejich spouštění.

5.6.5 Třída Aplication

Jedná se o hlavní třídu aplikace. Je to počáteční třída pro všechny další operace. Dědí ze třídy Thread za účelem jejího využití jako vlákna. Uchovává v sobě veškeré nastavení a informace o aktuálním stavu programu. Zajišťuje komunikace s instancemi tříd

[5] Jelikož všechny implementace MS podporovaly JMS specifikaci, můžeme zde pro ně používat označení JMS Provider (JMS server).



Obrázek 7: Schématické znázornění propojení aplikací

zajišťujícími odesílání, přijímání zpráv a správu uživatelů. Hlavní metodou je přepsaná metoda run(), která zajišťuje posloupnost prováděných funkcí – vygenerování uživatele, přihlášení uživatele, spuštění příjmu zpráv z topiků a front, cyklické náhodné generování „posílací operace“ (viz kapitola 5.3) a nakonec odhlášení uživatele. V závislosti na stanovené době běhu aplikace pak provede buď ukončení aplikace, anebo znovu opakování celého popsaného koloběhu.
Hlavní metody jsou:

- run()
- loginUser(), logoutUser() – metody pro přihlášení a odhlášení uživatele
- receivingMessages() – spuštění přijímaní zpráv z topiků a front
- sendDirectMessage(), sendDurableMassFile(), . . . – metody pro přípravu odesílání konkrétního typu zprávy
- další metody – např. pro generování uživatele, výběr dokumentu k odeslání, atd.

5.6.6 Třída MessagesProducer

Tato třída slouží k zajištění *odesílání zpráv na JMS server.* Komunikaci s konkrétním JMS serverem zajišťuje pomocí vytvořeného adaptéru. Probíhá v ní *vytváření připojení na server*, *následné vytváření session*, *destinací*, *zpráv*, *MessageProducerů*, *atd.* Probíhá zde veškeré nastavování technických parametrů komunikace a vlastností jednotlivých entit jako je

persistence, způsob potvrzování, priorita zpráv, nastavení MessageSelectorů, expirační dobu, atd. Nakonec pak realizuje samotné odeslání zprávy.

Toto všechno je zajišťováno pomocí několika metod, např. sendFileToDurableTopic(), sendMessageToTopic() a dalších.

5.6.7 Třída MessagesConsumer

Poskytuje obdobné chování jako třída MessagesProducer s tím, že zajišťuje *přijímání zpráv z JMS serveru*. Komunikace s konkrétním JMS serverem opět probíhá přes vytvořený adaptér. Probíhá v ní vytváření *připojení na sever*, *session*, *destinací*, *MessageConsumerů*, *vytváření jednotlivých instancí MessageListenerů* pro konkrétní typy destinací a jejich napojování na MessageConsumery. U toho bych zmínil, že jeden uživatel může například „naslouchat" na pěti dalších *FriendsDurableTopicích*. Existují pak dvě možnosti, jak se zachovat z hlediska MessageListenerů. Buď vytvořit pro každou tuto destinaci novou instanci MessageListeneru (v tomto případě MyMessageListener) nebo pouze jednu pro všechny. Po konzultaci na diskusním fóru jedné z vybraných implementací MS jsem zvolil druhou možnost.

Toto všechno je zajištěno pomocí několika metod, např. receivingFromReceiveQueue(), receivingFromTopic() a dalších.

5.6.8 Třída MyMessageListener a její potomci

Třída MyMessageListener poskytuje zobecněný přístup pro *asynchronní příjem zpráv z konkrétního JMS serveru*. Dědí ze třídy AMessageListener JMS Adaptéru, čímž je zajištěno asynchronní chování (viz kapitola 6. o JMS Adaptéru). Nutností je implementovat metodu processMessage(AMessage msg), která *zajišťuje příjem a následné zpracování zpráv.*

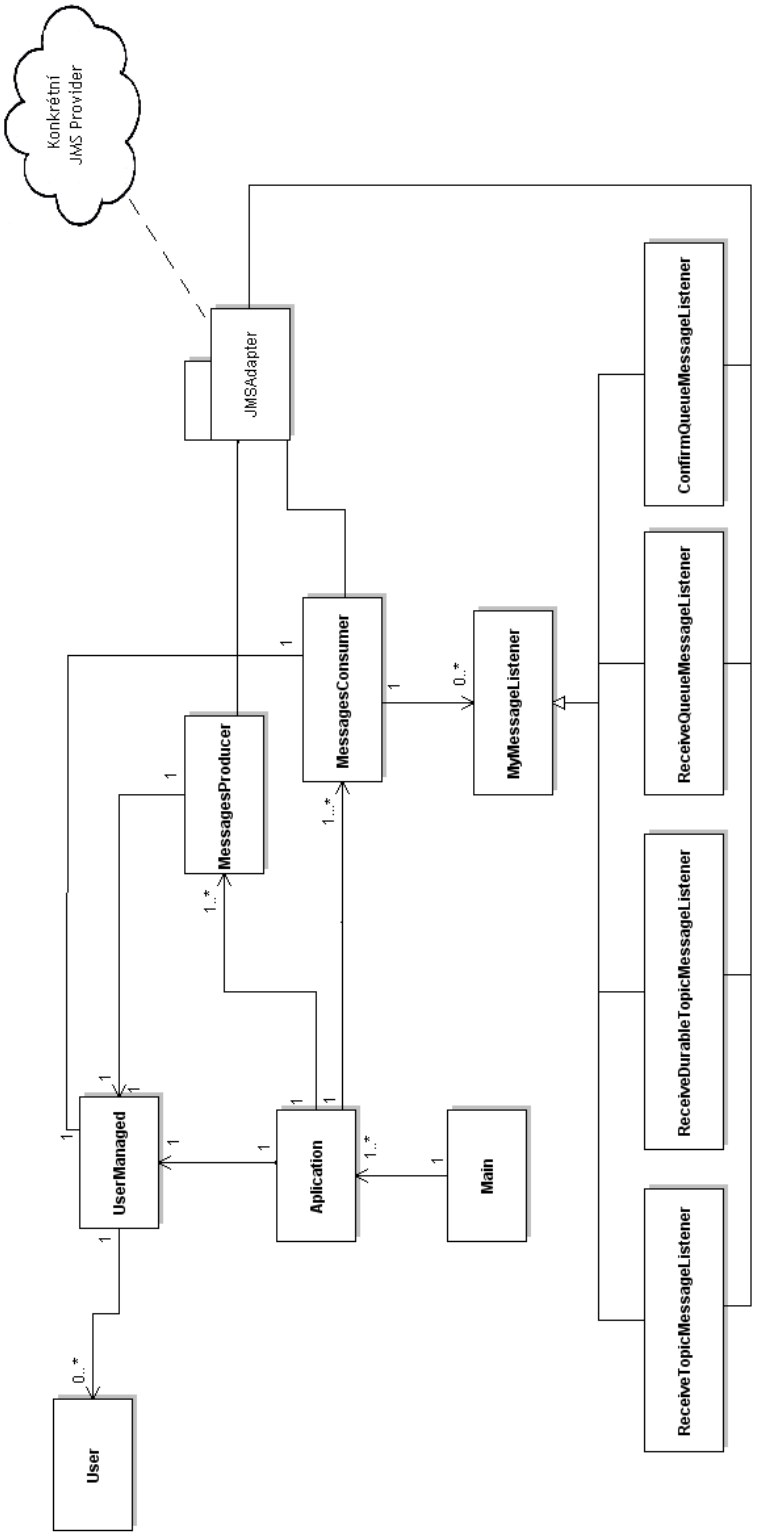
Potomci této třídy pro různé typy destinací jsou: ReceiveDurableTopicMessageListener, ReceiveTopicMessageListener, ReceiveQueueMessageListener, ConfirmQueueMessageListener

5.6.9 Třída User

Představuje *entitní třídu* reprezentující uživatele aplikace. Její vlastnosti jsou *id_user*, *name*, *login* a *userfriends* odpovídající tabulce User z databáze. Kromě get, set metod obsahuje přepsanou metodu toString () pro výpis informací o uživateli a metodu equals (...) pro porovnání dvou uživatelů.

5.6.10 Třída UserManaged

Třída pro práci s uživatelem a jeho relační mapování z a do databáze. Pro práci s databází je zde použit JDBC driver pro MySQL databázi. Kromě metod pro mapování (findUsers(), findUser (...) , existFriendShip (...) , UpdateLogin(...), atd.) jsou zde také *metody pro počáteční inicializaci aplikace*, tj. vytvoření uživatelů, jejich vazeb, vytvoření front a topiků a nastavení trvalých odběratelů.



Obrázek 8: Třídní diagram testovací aplikace

6 JMS Adaptér

6.1 Motivace

Různé implementace MOM poskytují různé API. Některé implementace poskytují pouze vlastní API. Velká většina podporující Javu ovšem implementuje specifikaci JMS, což umožňuje vyšší přenositelnost aplikace. Některé implementace poskytují JMS API i své vlastní API (jako například HornetQ). I přesto, že většina implementací podporuje JMS API 1.1, přidává k němu *svá různá rozšíření*. Tímto může být například velice rozdílný přístup k tvorbě připojení na server, tvorba front a topiků, rozšíření pro posílání objemných zpráv, atd.

Hlavním cílem mé diplomové práce bylo vybrat nejrozšířenější stávající implementace open source messaging systémů, provést jejich nasazení a analyzovat jejich výkonnost. Při nasazování se ukázalo, že i když všechny vybrané implementace MOM podporovaly JMS API 1.1, bylo nutno provést *vcelku velké změny do vyvíjené testovací aplikace*. Proto bylo dohodnuto, že bude vytvořen *adaptér* (pojmenovaný *JMS Adaptér*), který bude poskytovat skoro jednotný přístup pro různé implementace MOM.

Vybrané implementace MOM (JMS Provideři) pro analyzování jsou: *ActiveMQ*, *HornetQ*, *JORAM*, *OpenMQ*, *OpenJMS* (více viz kapitola 8), pro které byl vytvořen tento adaptér.

6.2 Struktura adaptéru

Vytvářený adaptér zaobaluje knihovny všech nasazovaných JMS Providerů tak, aby se k nim z aplikace mohlo přistupovat pouze a jen přes tento adaptér. V adaptéru jsem samozřejmě neimplementoval (nezaobaloval) všechny funkce a vlastnosti, které poskytují rozšířená API jednotlivých implementací nebo JMS API, ale pouze „hlavní jádro“ a funkce využívané v testovací aplikaci.

Jelikož všichni nasazovaní JMS Provideři podporují JMS specifikaci, struktura adaptéru je podobná struktuře JMS API.

Nyní budou popsány některé hlavní třídy adaptéru.

6.2.1 Třída Parameters

Obsahuje v sobě parametry pro jednotlivé JMS Providery, jako například URL serveru, přihlašovací údaje, parametry pro JNDI kontext a další.

6.2.2 Třída AContext

Někteří Provideři umožňují (vyžadují) vytvářet připojení pomocí JNDI. Tato třída slouží získávání instance Context a jeho zaobalení.

6.2.3 Třída AConnectionFactory

Tato třída slouží k zaobalení získávání jednotlivých ConnectionFactory pro různé JMS Providery, které jsou u různých implementací řešeny různě a odstranění nutnosti zadávat parametry připojení přímo v aplikaci. Je tedy defaultně nastaven *localhost*.

6.2.4 Třída ASession

Zaobaluje třídu Session JMS specifikace a realizuje její funkce s ohledem na způsoby její realizace u různých JMS Providerů, např. vytváření topiku (metoda createTopic(String name)).

6.2.5 Třída AAdminConsole

Někteří nasazovaní JMS Provideři poskytují speciální funkce na správu JMS serveru. Tyto funkce jsou dostupné přes tuto třídu a její metody.

6.2.6 Třídy AMessage, AMapMessage, ATextMessage a ABlobMessage

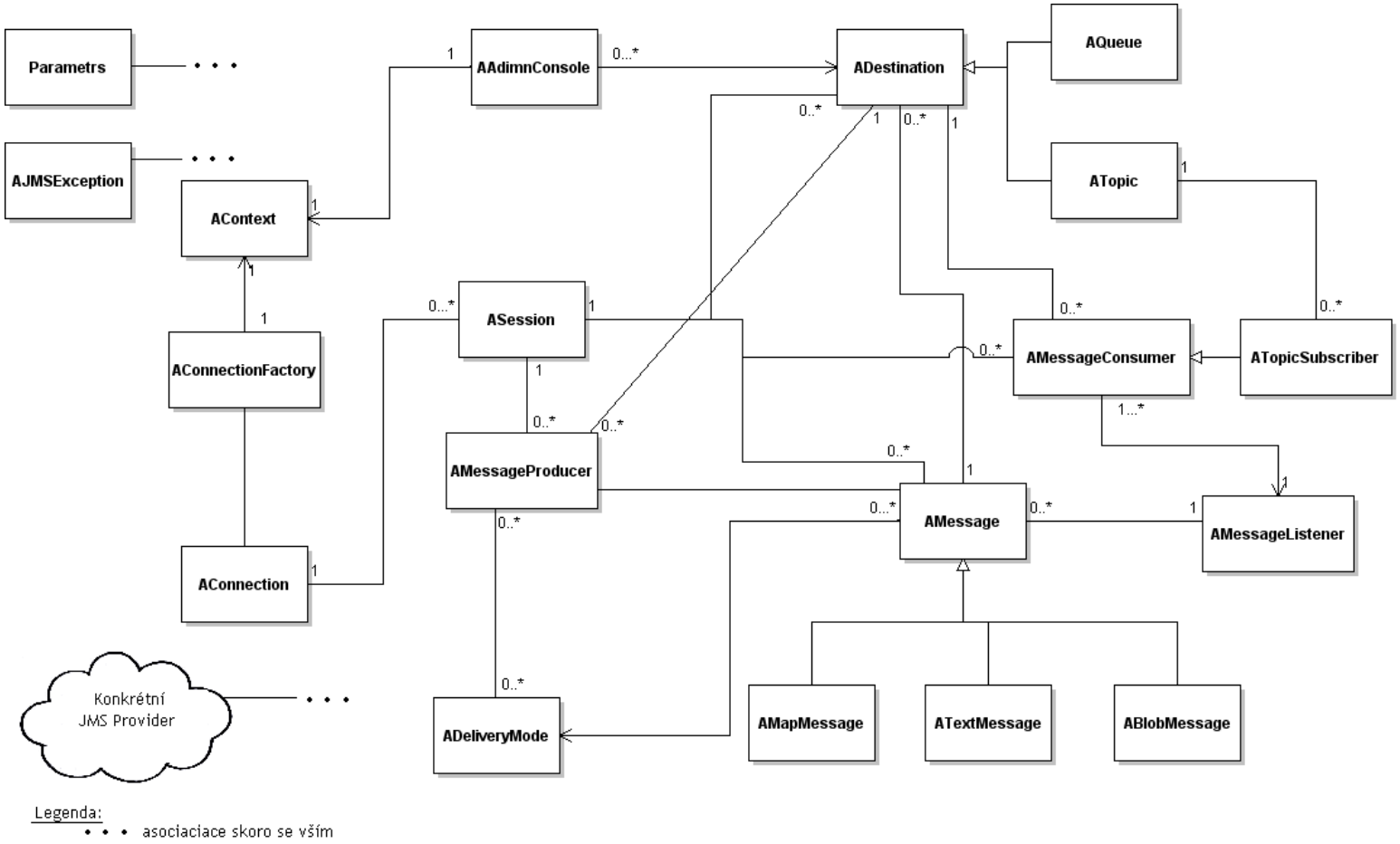
Třídy AMessege, AMapMessage a ATextMessage zaobalují jednoduchým způsobem jim odpovídající třídy JMS API.

ABlobMessage je třída reprezentující zprávu o velké velikosti, typicky elektronické dokumenty. Z implementovaných messaging systémů OpenJMS, OpenMQ a JORAM využívají pro tyto zprávy StreamMessage z JMS API, ActiveMQ má pro tyto zprávy vlastní ActiveMQBlobMessage a HornetQ využívá BytesMessage z JMS API. ActiveMQ i HornetQ sice umožňují využívat také StreamMessage, ale preferují své přístupy. Třída ABlobMessage zaobaluje tyto přístupy.

6.2.7 Třída AMessageListener

Třída implementující rozhraní MessageListener, které umožňuje asynchronní příjem zpráv. Přepisuje povinnou metodu onMessage(Message msg) za účelem zaobalení příjmů zpráv, které jsou rozdílně realizovány konkrétními implementacemi. Pro přepsání pak vystavuje abstraktní metodu progressMessage(AMessage message).

Celá statická struktura JMS adaptéru je zachycena na třídním diagramu na obrázku 9.



Obrázek 9: Třídní diagram JMS Adaptéru

7 Úprava aplikace pro testování

V předchozích několika kapitolách byla podrobně popsána naimplementovaná testovací aplikace. Tato aplikace musela být před prováděním výkonnostních experimentů *částečně upravena* a to zejména ze dvou důvodů:

Prvním důvodem bylo, že aplikace *simulovala „reálné" chování*, tj. uživatel je náhodně zvolen, posílá zprávu náhodně zvolenému počtu jeho přátel, náhodně je zvolen okamžik jeho odhlášení a přihlášení dalšího uživatele, atd. Tato *náhodnost* je ale u následných testů *zcela nežádoucí* a mohla by způsobovat *velice rozdílné výsledky*.

Druhým důvodem bylo, že aplikace měla víceméně pevně nastavené jednotlivé parametry, např. persistence a další. (viz. kapitola 5.5). Po provedení návrhu testů se ukázalo, že některé a které parametry by bylo vhodné nastavovat různě pro různé testovací případy.

Za účelem následného výkonnostního testování byly provedeny tyto změny:

- Vzájemné vztahy uživatelů byly upraveny, aby tvořily úplný graf (každý propojen s každým).
- Během celé doby běhu instance Aplikace bude přihlášen pouze jeden uživatel, jehož jméno bude předáváno přes příkazovou řádku.
- Byly odstraněny všechny příkazy Thread.sleep(...), za účelem dosažení maximálního vytížení serveru.
- Náhodné generování (uživatelů pro přihlášení, délky zprávy, odesílaného dokumentu, generování operace, atd.) bylo nahrazeno „pseudogenerátorem", který vracel hodnoty v pevně dané posloupnosti. Tento „pseudogenerátor" byl realizován třídou MyRandom a jejími metodami.
- Zprávy posílané do trvalých i netrvalých topiků byly posílány automaticky všem uživatelům – vypuštění filtrace zpráv.
- Kompletně byla předělána metoda run() třídy Aplication tak, aby umožňovala spouštění různých typů navržených testů. Byly přidány parametry nastavované přes příkazovou řádku pro výběr typu testů a operací, které mají být provedeny.
- Umožněno *nastavování různých parametrů* aplikace přes příkazovou řádku – persistence, Text- nebo Blob-Message, velikost přenášených dat, typ destinace, do které budou posílány zprávy, typ potvrzování, umožnit odeslání zpráv, přijímání zpráv, odesílání potvrzovacích zpráv, nastavení priority zpráv, doba expirace zpráv. Dále byl přidán také parametr pro zachování původního nastavení aplikace. Všechny tyto parametry byly zpřístupněny prostřednictvím třídy TestingParameters.
- Byly přidány *proměnné na počítání přenesených bytů a zpráv* a funkce zapisující naměřené hodnoty do souboru *results.txt*.
- Na základě výše uvedených změn došlo i k malým změnám v některých metodách.

8 Nasazování vybraných otevřených implementací MOM

Z výše popsaných otevřených existujících implementací MOM bylo vybráno celkem pět produktů. Tyto produkty byly nasazeny a byla s nimi provedena analýza výkonnostního testování. Výběr byl nakonec omezen pouze na produkty podporující JMS specifikaci. Výběr byl proveden především na základě jejich rozšíření a předpokládané výkonnosti. Vybrány byly tyto implementace (JMS Provideři): *ActiveMQ*, *HornetQ*, *OpenMQ*, *JORAM* a *OpenJMS*.

Nyní bude popsáno nasazování těchto implementací. Při nasazování jsem postupoval podle dostupných informací v dokumentacích jednotlivých poskytovatelů.

8.1 ActiveMQ

Nasazení ActiveMQ bylo ze všech vybraných implementací nejjednodušší a určitě uživatelský nejvíce přívětivé z mnoha důvodů. Pro nasazení byla použita v tu dobu *poslední stabilní verze 5.4.2*. ActiveMQ poskytuje jak zdrojové, tak binární soubory. V případě použití binárních souborů stačí stažené soubory rozbalit a pomocí dávkového souboru *activemq.bat* spustit server (ActiveMQ Message Broker). Defaultně server běží na portu 61616 a komunikace probíhá prostřednictvím TCP protokolu.

Pro správu serveru (správa destinací, atd.) poskytuje ActiveMQ velice přehlednou webovou konzoli na adrese *localhost:8161/admin*. Je poskytována také možnost správy přes JMX. Nastavení serveru je možné provádět přímo v XML souborech umístěných ve složce *conf* v hlavním adresáři.

Vytváření připojení na sever z aplikace bylo také snadné, jelikož ActiveMQ poskytuje možnost *přímého připojení bez nutnosti použití JNDI*, kterou jsem využil. Nicméně připojení s využitím JNDI podporuje také.

Při přenášení velkých souborů je nutné definovat *politiky přenosu*, což je nejvhodnější přímo v URL serveru. Další, na první pohled uživatelský příjemnou věcí bylo, že není nutno vytvářet *fyzické destinace* předem, ale ActiveMQ je vytvoří samo při prvním pokusu na ně něco poslat či z nich přijímat.

Při nasazování a práci ActiveMQ byla velkou výhodou přehledná a propracovaná dokumentace.

8.2 HornetQ

Nasazování HornetQ patřilo k těm obtížnějším. Pro nasazení byla použita *stabilní verze 2.1.2* ze srpna roku 2010, která je vestavěná v *JBoss Aplication Server 6 (AS6)*. HornetQ integrovaný v AS6 jsem využil zejména proto, že AS6 poskytuje *webovou administrační konzoli*, kde je možno spravovat i HornetQ server. Spuštění AS6 probíhá pomocí souboru *run.bat*. Server pak defaultně běží na portu 5445.

Konfiguraci HornetQ je možné částečně provádět přes administrační konzoli AS6, kde ovšem ze zvláštních důvodů (nějaký vada – bug) nejdou vytvářet fyzické destinace. Kompletní konfigurace jde provádět v konfiguračních souborech umístěných v adresáři

hornetq aplikačního serveru AS6. Jedná se především o *hornetq-configuration.xml* a *hornetq-jms.xml*.

U HornetQ je možné vytvářet své vlastní uživatele, kteří budou se serverem pracovat nebo je k dispozici defaultně uživatel *guest*. Tento uživatel ovšem nemá nastavená všechna práva, která například testovací aplikace potřebuje. Možností je *nastavit potřebná práva* nebo *zakázat zabezpečení* (tág *security-enabled*) v konf. souboru *hornetq-configuration.xml.*

Pro správu jsem využil zmíněnou administrační konzoli AS6 přístupnou na adrese *localhost:8080/admin-console*.

HornetQ nabízí kromě JMS API i své vlastní Core API. V aplikaci jsem použil JMS API, které je pak HornetQ nakonec stejně převádí na své Core API.

Jelikož HornetQ na rozdíl od ActiveMQ nevytváří fyzické destinace automaticky, bylo nutné je vytvářet ručně v konfiguračním souboru *hornetq-jms.xml*. U vytvářených front bylo potřeba pak nastavit ještě trvalost (tág *durable*), aby přežily i restart nebo pád serveru.

Kromě možnosti připojení se k serveru *pomocí JNDI* poskytuje také přímou možnost pomocí prostředků, které nabízí třída HornetQJMSClient. V aplikaci jsem použil možnost připojení bez JNDI.

Důležitou věcí, se kterou jsem se potýkal, je, že u HornetQ je potřeba si dávat pozor (víc než u jiných) na uzavírání vytvořených připojení (Connection), kdy neuzavřená připojení mohou způsobovat nestandardní chování celé aplikace.

8.3 OpenMQ

Pro nasazení OpenMQ byl zvolen *poslední dostupný release 4.5* z března roku 2011. OpenMQ umožňuje instalaci jak ze zdrojových, tak binárních souborů. V případě použití binárních souborů je pomocí GUI instalatéru nastaven server. Spuštění serveru pak probíhá pomocí souboru *imqbrokerd.exe*. OpenMQ server běží defaultně na portu 7676.

Pro správu serveru poskytuje OpenMQ jednoduchou ale přehlednou GUI aplikaci, která je spustitelná ze souboru *imqadmin.exe*. Pro rozsáhlejší správu je pak možné použít JMX API.

Pro připojení k serveru jsem využil pro OpenMQ klasickou možnost připojení přímo *bez využití JNDI*. Je možnost ale připojit se i užitím JNDI.

OpenMQ podporuje čistou specifikaci JMS a neposkytuje například vlastní třídy pro přenos velkých souborů.

Stejně jako u ActiveMQ není nutné vytvářet předem fyzické destinace. O to se postará JMS server při prvním pokusu o připojení k nim.

U OpenMQ je dokumentace méně přehledná a těžko se v ní hledají potřebné informace (alespoň teda z mého pohledu).

8.4 JORAM

Nasazování JORAM implementace patří k těm nejsložitějším. Pro nasazení byla vybrána poslední dostupná verze *JORAM 5.7.0.* Pro stažení binárních souborů je potřeba zadat své jméno a email a souhlasit s licenčními podmínkami. Po stažení stačí komprimované soubory rozbalit. Spouštěcím souborem je pak pro Windows *single_server.bat* nacházející

se v adresáři *samples/bin*. Pro nastartování serveru je nutno nastavit systémové proměnné *JAVA_HOME* a *JORAM_HOME*. JORAM JMS server defaultně běží na portu 16400.

Veškerou konfiguraci serveru je možno provádět v XML konfiguračních souborech nacházejících se v adresáři *samples/config*. V těchto souborech je potřeba změnit nastavení proměnné *Transaction*. Její defaultní nastavení je *fr.dyade.aaa.util.NullTransaction* a je ho potřeba změnit na *fr.dyade.aaa.util.NTransaction*, aby bylo možno posílat persistentní zprávy.

Pro administraci JORAM nabízí *webovou konzoli*, kterou je prý možno podle tutoriálu zprovoznit z balíčků přiložených u instalace serveru. Mnou stažený poslední release tyto balíčky neobsahoval. Druhou poskytovanou možností je částečná správa pomocí JMX. S využitím programu *Visual VM 1.3.3* jsem použil tedy tuto možnost správy serveru. Tento způsob ale není příliš uživatelsky přívětivý.

JORAM obdobně jako HornetQ nevytváří fyzické destinace automaticky. Bylo tedy nutné je vždy vytvářet ručně. K tomuto vytváření jsem raději, místo užití JMX prostřednictvím Visual VM, využil poskytované administrační API. Pomocí něj bylo pak možno nastavovat třeba i to, zda je fronta jen pro čtení nebo i zápis pro jednotlivé uživatele, vytváření uživatelů, aj.

JORAM na rozdíl od předchozím JMS Providerů *neposkytuje* možnost *přímého připojení bez JNDI*. Bylo tedy nutné použít rozdílného způsobu připojení než předchozích případech. K zajištění připojení bylo nutné nejdříve vytvořit prostřednictvím administračního API v jmenném prostoru *ConnectionFactory (cf)* a *uživatele (anonymous)*, kteří byli následně využíváni v aplikace.

8.5 OpenJMS

Jako poslední jsem pro nasazení zvolil OpenJMS. Přesto, že jeho vývoj byl ukončen v roce 2006 a poskytuje pouze nejzákladnější funkce (viz kapitola 4), zdálo se mi, že je poměrně známý a možná i rozšířený. Pro svoji jednoduchost je možná oblíbený pro použití v jednoduchých aplikacích. A také bude zajímavé porovnání ostatních MS s produktem, jehož vývoj se zastavil před přibližně 5–6 lety.

Pro nasazení jsem použil poslední dostupný release *OpenJMS 0.7.7-beta-1*. Po jeho stažení stačilo archív rozbalit. Spuštění serveru se pak provádí pomocí souboru *startup.bat*. Pro správný chod serveru je nutno mít nastavenou systémovou proměnnou *JAVA_HOME*. Je možné, ale není vyžadováno, nastavit i *OPENJMS_HOME*. OpenJMS server defaultně běží na portu 3535 a komunikace probíhá pomocí defaultně nastaveného *TCP protokolu*.

Veškerou konfiguraci je možno dělat přímo v konfiguračním souboru *openjms.xml* v adresáři *config* hlavní adresářové struktury nebo použít *administrační API*, které jsem využil já. OpenJMS poskytuje také jednoduchou GUI aplikaci pro správu vytvářených destinací.

Pro připojení aplikace k serveru bylo využito JNDI kontextu, jelikož OpenJMS jinou možnost neposkytuje. Aby mohlo dojít k vytvoření připojení, je nutné mít vytvořeného *uživatele* a *ConnectionFactory* v *openjms.xml*. Defaultně již ale existuje uživatel *admin* a jedna vytvořená ConnectionFactory, které je možné využít.

Jelikož OpenJMS obdobně jako HornetQ nebo JORAM nevytváří destinace automaticky při jejich prvním volání, je nutné je vytvořit ještě před spuštěním aplikace. K tomu jsem využil zmíněné administrační API.

Oproti předchozím JMS Providerům je u OpenJMS nutné jiným způsobem vytvářet instance již fyzicky existujících front a topiků a to s pomocí JNDI kontextu. Standardní vytváření pomocí instance třídy Session, které definuje JMS specifikace, je sice možné, ale přináší nestandardní chování.

Nasazování OpenJMS patřilo k jednodušším i proto, že dokumentace poskytovaná na jejich stránkách přehledná a srozumitelná.

9 Výkonnostní testování vybraných implementací MOM

Hlavním cílem diplomové práce bylo provedení výkonnostních testů či experimentů s jednotlivými vybranými implementacemi messaging systémů. V této rozsáhlé kapitole budou v úvodu zmíněna některá existující volně dostupná porovnání MS. Dále pak bude popsána metodika testování, volba testovacích dat, atd. A nakonec budou popsány jednotlivé prováděné experimenty, jejich výsledky a zhodnocení.

9.1 Existující porovnání

Před prováděním návrhů testů a jejich vykonáváním jsem se pokusil nalézt ve zdrojích dostupná *existující výkonností porovnání implementací MOM*, a to především zaměřená na otevřené produkty. Z těchto porovnání jsem se chtěl jednak inspirovat pro návrh vlastních testů a také seznámit s výsledky, které byly v těchto testech zjištěny. Na první pohled bylo nalezeno relativně velké množství odkazů. Při bližším zkoumání jsem ale zjistil, že existujících volně dostupných výkonnostních porovnání není příliš mnoho. Navíc většina je relativně starého data, což již není moc vypovídající a většinou zaměřené na komerční produkty. Pro zmínku byly vybrány 4 porovnání. První z nich je zaměřené na komerční produkty a další tři pak na open source produkty z posledních tří let.

9.1.1 Porovnání od Fiorano

Nejkvalitnější a nejaktuálnější nalezené porovnávaní většinou komerčních produktů je porovnání zveřejněné firmou Fiorano, která vyvíjí svůj komerční messaging systém FioranoMQ (viz kapitola 4). Fiorano zveřejňuje porovnání aktuálně nejlepších MS asi přibližně každé 2 roky. Toto porovnání je z roku 2011.

Porovnávány byly kromě komerčních produktů *FioranoMQ 9.3.1*, *Sonic MQ 7.6*, *Tibco EMS v4.4.0*, *IBM WebSphere MQ 7.0* i nekomerční produkty *ActiveMQ 5.3.0*, *Sun MQ 4.3* a *JBoss Messaging 1.4.4*.

Byly testovány dva klasické modely použití topiku a to:

- *Nepersistentní odesílatel a netrvalý odběratel* (Non-Persistent Publishers & Non-Durable Subscribers)
- *Persistentní odesílatel a trvalý odběratel* (Persistent Publishers & Durable Subscribers)

Na prvním modelu pak byly provedeny testy na *topikovou škálovatelnost, serverovou škálovatelnost a případ, kdy je jeden topik, jeden odesílatel a několik odběratelů.*

Pro druhý model pouze *případ, kdy je jeden topik, jeden odesílatel persistencích zpráv a několik trvalých odběratelů.*

V každém testovacím případu byly postupně navyšovány počty topiků, odesílatelů a odběratelů. A to na hodnoty 1, 10, 25 a 50.

Pro testování zde byly použity dva servery (jeden pro klienty, druhý pro server). Všichni provideři běželi v defaultním nastavení.

Jako testovací metodologie a řízení generování zátěže byla zde použita knihovna tříd *Sonic Test Harness*, vyvíjená od roku 2007, která poskytuje prostředky pro generování zátěže a konfiguraci jednotlivých parametrů testů.

Ve všech provedených testech je s přehledem nejvýkonnějším produktem *FioranoMQ*, kde své konkurenty v některých testech překonal několikanásobně. S velkým odstupem za FioranoMQ je ve většině případů *Sonic MQ*. Pro některé provedené testy (např. topiková škálovatelnost) dosahují dobrých výsledků i open source produkty *ActiveMQ* a *JBoss Messaging*. Jednoznačně nejhůře dopadly výsledky těchto testů pro *IBM WebSphere MQ*.

Detailní informace o provedeném porovnání, použité metodologii, testovacích scénářích, testovacím prostředí, topologii a výsledcích lze pak nalézt na [30].

9.1.2 Porovnání tří nekomerčních produktů z roku 2008

Toto porovnání je z února 2008. Zveřejnila ho společnost C2B2 zabývající se systémovou integrací.

Porovnávány byly pouze open source JMS implementace a to: *JBoss MQ 4.2.2*, *JBoss Messaging 1.4.0* a *OpenMQ 4.1*. Jelikož JBoss MQ i JBoss Messaging se již dále nevyvíjí, ztrácí toto porovnání částečně na aktuálnosti.

Testování zde bylo zaměřeno pouze na odesílání zpráv a to pouze do front. Testování bylo zaměřeno především na tři oblasti:

- *Doba potřebná k navázání spojení se serverem*
- *Doba potřebná pro odeslání jedné zprávy* – bylo vytvořeno připojení na server a odesláno 1, 10, 100, 1000 nebo 10000 zpráv a následně uzavřeno připojení.
- *Charakteristiky selhání* – množství zpráv, které je možno poslat na server než dojde k chybě nebo selhání serveru a chování serveru při selhání.

Byly použity dva servery (jeden pro klientské aplikace, druhý pro server). Odesílané zprávy byly textové zprávy (TextMessage) o velikosti 30 znaků. Test byl opakovaně proveden 5x. JMS servery byly ponechány v defaultním nastavení.

Pro první testovací případ měl nejlepší výsledky *JBoss MQ*, který nejrychleji navazoval připojení na server.

U druhého testovacího scénáře byly výsledky rozdílné v závislosti na počtu posílaných zpráv. *JBoss MQ* poskytoval lepší výsledky pro menší počet poslaných zpráv najednou, pro velké množství zpráv pak byl rychlejší *JBoss Messaging*.

U testu zabývající se selháním při neustálém odesílání zpráv na server bylo zjištěno, že nejdéle vydržel přijímat zprávy *JBoss MQ*. Na druhou stranu ale u *Open MQ nedošlo k úplnému pádu serveru*, jen zamezil odesílání další zpráv klientem. U *JBoss MQ* a *JBoss Messagingu* došlo *k pádu celého serveru*.

Toto porovnání se bohužel vůbec nezaměřovalo například na persistenci zpráv, transakce a způsoby potvrzování, topikovou nebo serverovou škálovatelnost, atd. Další podrobnosti lze nalézt na [39].

Tests on Development server - single instance - no replication		
	Average number of messages per seconds during 5 min test	
Test name	**ActiveMQ**	**HornetQ**
1KB Text message, no persistence, no transaction, 25 senders, 25 receivers	26207.58	28655.23
10KB Text message, no persistence, no transaction, 25 senders, 25 receivers	9037.4	5606.64
100KB Text message, no persistence, no transaction, 25 senders, 25 receivers	720.91	565.58
300KB Text message, no persistence, no transaction, 25 senders, 25 receivers	222.3	181.51
300KB BYTES message, no persistence, no transaction, 25 senders, 25 receivers	747.3	1126.66
1KB Text message, persistence, transaction, 25 senders, 25 receivers	35.68	5314.02
10KB Text message, persistence, transaction, 25 senders, 25 receivers	27.08	2723.46

Tests on Development server - single active instance - replication to passive node		
Note only persistent tests are done in this setup: HornetQ does not replicate non persistent messages.		
	Average number of messages per seconds during 5 min test	
Test name	**ActiveMQ**	**HornetQ**
1KB Text message, persistence, transaction, 25 senders, 25 receivers	38.37	4391.62
10KB Text message, persistence, transaction, 25 senders, 25 receivers	27.72	1884.46

Obrázek 10: Výkonnostní porovnání ActiveMQ vs. HornetQ z roku 2011; zdroj: [40]

9.1.3 Porovnání ActiveMQ vs. HornetQ z roku 2011

Zajímavé nezávislé porovnání dnes asi nejrozšířenějších a nepropracovanějších otevřených implementací MS *ActiveMQ* a *HornetQ* je k dispozici na [40].

Také u tohoto testu autoři použili framework *Sonic test herness* umožňující jednoduché generování zátěže na JMS servery.

Testování bylo realizováno v roce 2011 a omezuje se pouze na práci s frontami.

Jednotlivé testovací scénáře jsou testováním výkonnosti jednotlivých messaging systémů pro různé kombinace nastavení velikosti zprávy (1KB, 10KB, 100KB, 300KB), TextMessage nebo BytesMessage, persistentní nebo nepersistentní a přístup s použitím transakcí nebo přístup bez transakcí.

Výsledky testů jsou vidět na obrázku 10 převzatého z webových stránek autora testování [40].

Výsledky testů ukazují, že pro nepersistentní textové zprávy ActiveMQ i HornetQ poskytují přibližně stejné výsledky. Pro 300KB zprávu již HornetQ poskytuje lepší výsledky než ActiveMQ. Obrovský rozdíl pak ale nastává pro persistentní zprávy s transakčním potvrzováním, kdy HornetQ je přibližně 100x rychlejší než ActiveMQ.

Test se také zabýval výkonností serveru, v případě, že je vytvářena kopie (replikace) zpráv na dalším serveru. Zde také výsledky hovoří jasně pro HornetQ.

Bohužel tento test se vůbec nezabývat prací s topiky, trvalými odběrateli, serverovou nebo topikovou škálovatelností, aj.

9.1.4 Rozsáhlé porovnání čtyř nekomerčních produktů z roku 2011

Velice komplexní nezávislé porovnání implementací messaging systémů založených na *STOMP protokolu* je dostupné z [41]. Výsledky tohoto porovnání byly zveřejněny v prosinci 2011. Testovány byly tyto produkty: *ActiveMQ*, *HornetQ*, které podporují JMS a dále pak *RabbitMQ* a *ActiveMQ Apollo (Apollo)*.

Testovací scénáře byly tvořeny kombinací následujících možností:

- Topik nebo fronta
- 1, 5, 10 odesílatelů, odběratelů, destinací
- 20B, 1KB, 256KB zprávy
- persistentní, nepersistentní zprávy

Test běžel na dvojici různých strojů a doba trvání jednoho testovacího případu byla 15 sekund.

Testovací případy byly rozděleny do několika kategorií:

- *Výkonnost při posílání do topiku bez připojených trvalých odběratelů* – testováno bylo počet přenesených zpráv při jejich různých velikostech. Nejlepších výsledků dosahoval Apollo a pro velké zprávy také HornetQ.
- *Výkonnost při posílání do fronty* – porovnávání výkonnosti pro odesílání persistentních a nepersistentních zpráv. Pro persistentní přenos dosahoval výborných výsledků HornetQ a Apollo, pro nepersistentní pak RabbitMQ a Apollo.
- *Výkonnost při přijímání z fronty* – Apollo byl zde jednoznačně nejlepší MS. HornetQ byl mnohem rychlejší než ActiveMQ.
- *Testy na topikovou a serverovou škálovatelnost* – zde byly testovány všechny možné kombinace výše uvedených možností.

Velice rozsáhlé výsledky a grafy tohoto porovnání jsou zveřejněny na stránkách autora tohoto porovnání na [41].

Toto velice rozsáhlé porovnání poskytuje pohled na výkonnost jednotlivých testovaných MS snad ze všech možných úhlů. Z výsledků vyplývá, že asi nejvýkonnějším produktem je nově se vyvíjející ActiveMQ Apollo. Ukazuje se zde také, že zbylé tři MS (ActiveMQ, HornetQ, RabbitMQ) jsou si prakticky rovny a záleží na konkrétní situaci použití. Nelze tak na základě tohoto porovnání určit jednoznačné pořadí výkonnosti.

9.2 Metodika testování

Pro návrh testovacích případů a metodiky testování jsem se inspiroval z velké části existujícími porovnáními nalezenými na webu. Navržené testy, které jsou pak detailně popsány v další kapitole, jsou zaměřeny především na *výkonnost (propustnost)* jednotlivých MS v různých situacích. Kromě výkonnostního zhodnocení, zde bude ale také zmíněno

zhodnocení *stability* jednotlivých JMS serverů, které se dalo vypozorovat při provádění výkonnostních testů.

Veškeré prováděné testy byly realizovány na vytvořené *testovací aplikaci*, která byla detailně popsaná v kapitole 5. Aby bylo možné testy provádět, byla původně naimplementovaná aplikace částečně upravena, což bylo popsáno v kapitole 7.

Při provádění testů byl měřen *počet přenesených zpráv*, případně *počet přenesených bajtů*. Doba běhu každého testovacího scénáře byla *stanovena na 60 sekund*. Aby se zamezilo možné chybě při testování, byly testy *provedeny vícekrát*. Z důvodů toho, že celková doba provedení všech testovací scénářů jedenkrát trvala asi cca 20 hodin, byly provedeny některé testovací scénáře *3x*, některé pouze *2x*.

Testy byly prováděny na počítačové sestavě s touto *hardwarovou konfigurací*[6]: procesor Intel® Core2 DuoTM T5270 1,4GHz, operační paměť o velikosti 2GB a grafickou kartou ATI Mobility RadeonTM HD 2400 XT s pamětí 256 MB. Operační systém počítače byl 32 bitový Microsoft Windows XP Service Pack 3 a Sun JDK 1.6.0-24.

Další testovací podmínky byly:

- Během testů na počítači nebylo nic jiného spuštěno.
- Po každém provedení testovacího scénáře byly vyprázdněny destinace.
- Po každém provedení testovacího skriptu (sada testovacích případů) byl restartován počítač a nově vytvořeni uživatelé i jejich destinace.
- Z podstaty navržené testovací aplikace bylo pro každou odesílanou zprávu vytvořeno nové připojení. Pro příjem zpráv bylo jedno připojení po celou dobu běhu (viz také kapitola 5.5).
- Bylo zvoleno *defaultní nastavení* testovaných MS s výjimkou JORAM, kde muselo být povoleno posílání persistentních zpráv. Dále pak se ukázalo nutné provést navýšení dostupné paměti pro servery OpenMQ, JORAM a OpenJMS, kde byla paměť navýšena na 1024MB (víc nebylo možno na tomto počítači navýšit).

Na základě existujících porovnání a vlastních nápadů byly jednotlivé testovací scénáře tvořeny kombinacemi několika parametrů a nastavení.

Tyto parametry jsou:

- Fronta, topik nebo topik s trvale připojenými odběrateli (trvalý topik)
- 20B, 5KB textové zprávy, 512KB, 2MB nebo 4MB zprávy ve formě el. dokumentů
- TextMessage, BlobMessage, případně MapMessage
- Persistence
- Transakce nebo netransakční potvrzování v režimu *DUPS_OK_ACKNOWLEDGE* (nejméně náročný potvrzovací režim)

[6] Ideálním řešením by bylo, kdyby server i klienti běželi na různých strojích

- 1, 5 nebo 15 odesílatelů (Producer)
- 1, 5 nebo 15 odběratelů (Consumer)
- Různé počty destinací v závislosti na konkrétním testu
- Odesílání zpětných potvrzovacích zpráv (Confirm Message)

9.3 Realizace testování a výsledky

Navržené experimenty byly rozděleny do *pěti částí (kategorií)* s ohledem na jejich typ a zaměření. První kategorie je zaměřena na výkonnost MS na celé aplikaci, „se vším všudy“. Druhá kategorie je pak zaměřena na konkrétní činnosti a jejich různá nastavení (posílání pouze textových persistentních zpráva s transakčním potvrzováním, atd.). Třetí část je zaměřena pouze na odesílání zpráv do topiků a front, čtvrtá je pak zaměřena na serverovou a topikovou škálovatelnost. Poslední část je věnována rychlosti vytváření připojení na server.

Pro každou tuto kategorii (kromě poslední) byl vytvořen *testovací skript*, aby bylo zajištěno částečné zautomatizování provádění testů.

Před provedením testů se dalo očekávat, že nejlepších výsledků by měl, podle dostupných informací dosahovat *HornetQ*, následovaný pravděpodobně *ActiveMQ*, následně pak *OpenMQ* a *JORAM*. Nejhorší výsledky se daly očekávat od již dnes dále nevyvíjeného *OpenJMS*.

9.3.1 Výkonnostní testy na celé aplikaci

Protože aplikace byla navržena tak, aby realizovala nějakou „smysluplnou reálnou“ činnost, bylo by vhodné *otestovat jednotlivé messaging systémy přímo na této aplikaci* s jejím *původním nastavením*. MS budou na aplikaci testovány jako kdyby běžela v reálném provozu. To například znamená, že budou prováděny všechny navržené funkce, tj. budou posílány potvrzovací zprávy jako odpovědi na přijaté zprávy z fronty nebo trvalého topiku, různé priority pro různé typy zpráv, atd. Na základě tohoto experimentu se ukáže, která z vybraných implementací MS by byla pro tuto aplikaci *nejvhodnější*.

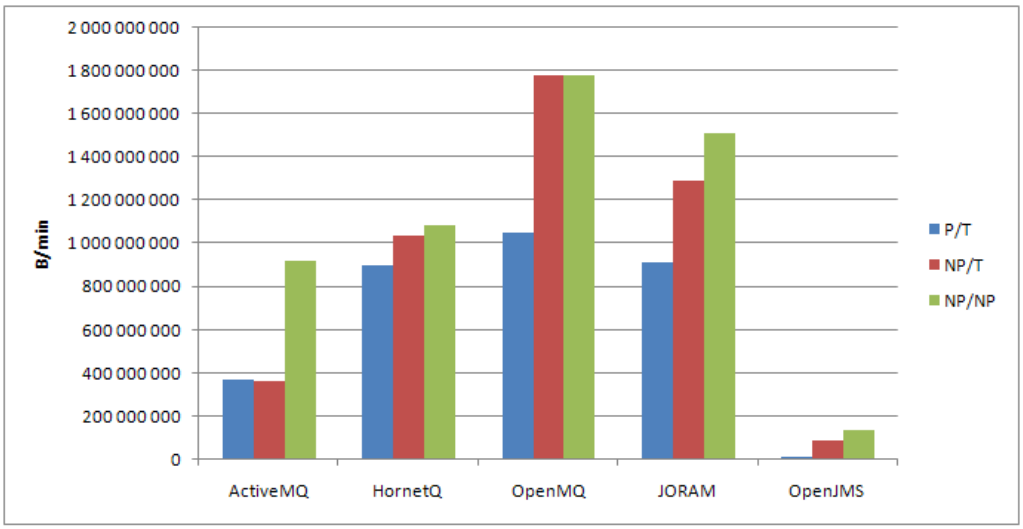
Tento test bude realizován pomocí tří testovacích případů. Bude zde porovnáván *persistentní versus nepersistentní způsob nakládání se zprávami a potvrzování pomocí transakcí versus potvrzování bez transakcí s použitím Dups_ok_acknowledge režimu*, který je založen na způsobu „líného potvrzování“.

Testovací případy jsou:

1. persistentní/transakciovaný model (P/T)
2. nepersistentní/transakciovaný model (NP/T)
3. nepersistentní/netransakciovaný model (NP/NT)

	P/T	NP/T	NP/NT
ActiveMQ	367 775 411B	363 452 790B	919 467 639B
HornetQ	897 330 660B	1 039 846 943B	1 087 987 227B
OpenMQ	1 051 112 648B	1 780 898 799B	1 783 409 206B
JORAM	916 696 850B	1 288 850 309B	1 510 858 055B
OpenJMS	10 364 493B	91 876 450B	136 805 280B

Tabulka 1: Výsledky výkonnostních testů na celé aplikaci



Obrázek 11: Graf s výsledky výkonnostních testů na celé aplikaci

Test byl prováděn s *pěti uživateli* (5 JMS klientů), kde každý byl jak odesílatel, tak odběratel. Každý testovací případ byl proveden *3 krát nezávisle na sobě*. Výsledek byl stanoven jako průměr získaných hodnot. Metrikou zde byly *přenesené bajty*.

Výsledky testů je možné vidět v tabulce 1 a grafu na obrázku 11.

Z výsledků testů vyplývá, že jednoznačné nejhorším produktem se jeví *OpenJMS*, který je několikanásobně pomalejší než ostatní MS. U persistentního případu ovšem jsou výsledky částečně zkreslené tím, že OpenJMS neumožňuje přenos persistentních zpráv větších než 32KB. Proto byly přes OpenJMS posílány jen persistentní zprávy o velkosti 20B a 5KB.

Pro přenos persistentních zpráv se ukázalo, že OpenMQ, JORAM i HornetQ poskytují přibližně stejné výsledky okolo *1000MB přenesených dat za minutu*. Víc než dvojnásobně pomalý byl pak ActiveMQ, což je vcelku překvapení.

Pro přenos nepersistentních zpráv jsou pak výsledky s výraznějšími rozdíly. Nejlepší propustnost zde poskytuje *OpenMQ*, které následuje JORAM, HornetQ a ActiveMQ.

Velmi zajímavé je porovnání persistentního a nepersistentního přenosu u jednotlivých MS. Kdy u ActiveMQ se zdá, že to, jestli je zpráva persistentní nebo ne, nehraje žádnou roli. Naopak u OpenMQ jsou nepersistentní zprávy přenášeny mnohem rychleji.

Nakonec byl porovnán přenos s transakčním a bez transakčního potvrzování. Zde je vidět u ActiveMQ *obrovský nárůst přenesených dat* při netransakčním potvrzování. Částečný nárůst můžeme vidět také u HornetQ, JORAM nebo OpenJMS. Naopak OpenMQ má stejné výsledky jak u transakčního, tak netransakčního přenosu.

9.3.2 Výkonnostní testy vybraných vlastností

V předchozím „celkovém" testování byly pro porovnání zahrnuty pouze atributy persistence a transakce. Tato druhá kategorie experimentů je zaměřena na *konkrétní typy (situace)* použití messaging systémů. Testovací případy jsou zde tvořeny kombinací *persistence*, *typu potvrzování* (transakční nebo Dups_ok_acknowledge režim), *různé velikosti zpráv* (20B, 5KB, 512KB, 2MB a 4MB), *typem destinace* (fronta, topik nebo trvalý topik) a *typem posílaných zpráv* (TextMessage nebo BlobMessage). V každém testovacím případu budou odesílány a přijímány pouze zprávy konkrétního typu o pevně dané velikosti, aby bylo možno měřit počet přenesených zpráv za dobu běhu testovacího případu. To je rozdíl také oproti předchozí kategorii, kde byly posílány také např. prázdné potvrzovací zprávy, atd. Cílem tohoto typu testů je *stanovit nejlepší messaging systém pro různé situace*. Například některý messaging systém může mít lepší výsledky pro malá data, jiný pro přenášení velkých dat, jiný pro persistentní přenos, atd.

Test byl prováděn s *pěti uživateli* (5 JMS klientů), kde každý je jak odesílatel, tak odběratel. Každý testovací případ byl proveden *2 krát* nezávisle na sobě. Výsledek byl stanoven jako průměr získaných hodnot. Metrika zde byla *počet přenesených zpráv.*

Celkem bylo navrženo 9 testovacích případů, kde každý je prováděn pro 5 různých velikostí přenášených dat (20B, 5KB, 512KB, 2MB a 4MB). Celkem tedy 45 testovacích případů.

Tyto testovací případy jsou:

1. persistentní/transakciovaný model pro práci s frontou (Q/P/T)
2. nepersistentní/transakciovaný model pro práci s frontou (Q/NP/T)
3. nepersistentní/netransakciovaný model pro práci s frontou (Q/NP/NT)
4. persistentní/transakciovaný model pro práci s topikem (T/P/T)
5. nepersistentní/transakciovaný model pro práci s topikem (T/NP/T)
6. nepersistentní/netransakciovaný model pro práci s topikem (T/NP/NT)
7. persistentní/transakciovaný model pro práci s trvalým topikem (DT/P/T)
8. nepersistentní/transakciovaný model pro práci s trvalým topikem (DT/NP/T)
9. nepersistentní/netransakciovaný model pro práci s trvalým topikem (DT/NP/NT)

Kde 20B a 5KB zprávy jsou realizovány jako TextMessage. Zprávy o velikosti 512KB, 2MB a 4MB jsou pak předem připravené soubory různých typů posílané jako BlobMessage.

Jelikož výsledky tohoto experimentu jsou dost rozsáhlé, byla většina tabulek a grafů zařazena do příloh. Na ukázku zde byly zařazeny tabulky 2, 3, 4, které se týkají práce s frontou a k nim odpovídající graf na obrázku č. 12. Tabulky 12, 13, 14, 15, 16, 17 a grafy na obrázcích 18 a 19 týkající se topiku a trvalého topiku, jsou zařazeny v přílohách.

Z výsledků je patrné mnoho zajímavých pozorování, která jsou nejlépe patrná na přiložených grafech.

Některé plynoucí závěry z výsledků těchto testů:

- U posílání a přijímání *menších zpráv* dosahovali nejlepších výsledků OpenMQ, HornetQ a také velmi překvapivě OpenJMS. Docela slabé výsledky pak poskytovali ActiveMQ a JORAM. Ovšem u zpráv o větší velikosti se výsledky srovnávaly a to zejména pro persistentní zprávy – nejlépe se zde jeví JORAM, OpenMQ a HornetQ.

- Z výsledků je také patrné, že ActiveMQ příliš nerozlišuje, zda je posílána persistentní nebo nepersistentní zpráva. Naměřené hodnoty jsou si prakticky rovny. Velký rozdíl ovšem je, když dojde k posílání/přijímání bez transakčního potvrzování, zde jsou výsledky až 5x lepší (hlavně pro malá data) a ActiveMQ patří společně s HornetQ k nejvýkonnějším.

- U HornetQ, JORAM, OpenJMS a OpenMQ pak je znatelný větší či menší nárůst počtu přenesených zpráv v *nepersistentním režimu*. Především u OpenMQ pro posílání BlobMessage je nárůst značný. Naopak rozdíl v typu potvrzování se výrazněji neprojevuje.

- OpenJMS podává velice dobré výsledky pro práci s *TextMessage o malé velikosti*, pro posílání rozsáhlých dat jsou výsledky velice slabé.

- Důležitým pozorováním je, jak se MS staví k *persistenci u netrvalého topiku*. Z principu JMS specifikace totiž nemá persistence žádný význam, tudíž by výsledky persistentních a nepersistentních testů měly být stejné. S výjimkou OpenJMS, kde výsledky nepersistentního testu jsou vyšší, jsou si výsledky víceméně rovny, což odpovídá předpokladům.

- U implementace JORAM, na rozdíl od ostatních, je vidět vcelku vyrovnané hodnoty pro všechny velikosti přenášených dat (díky tomu se řadí k nejvýkonnějším MS pro velká data). Naopak pro některé ostatní (např. ActiveMQ pro NP/NT) docela výrazně klesá počet přenesených zpráv při zvětšující se velikosti zprávy.

- Pro *trvalý topik s nepersistentními zprávami* dosahuje nejlepších výsledků HornetQ. Ale pro persistentní model má mnohem horší výsledky.

- Obecně by se dalo říct, že nejlepší výsledky podává HornetQ a to především u menších dat (20B, 5KB, 512KB). Také OpenMQ podává také velice dobré výsledky a především pro veliká data má nejlepší hodnoty. ActiveMQ je společně s HornetQ nejvýkonnějším MS pro netransakciovaný přenos. Avšak použití transakcí dramaticky snižuje výkon ActiveMQ. JORAM naopak podává velice slušné výsledky pro velká data, ale pro malé zprávy není příliš výkonný.

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	1641	5820	6038	1585	3960
TextMessage (5KB)	1852	3247	2547	1076	1880
BlobMessage (512KB)	802	2731	1416	1002	-
BlobMessage (2MB)	287	383	416	416	-
BlobMessage (4MB)	171	210	192	230	-

Tabulka 2: Výsledky výkonnostních testů pro testovací případ Q/P/T

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	1764	7896**	6552	1308	5010*
TextMessage (5KB)	1657	4506	2900	1202	2126
BlobMessage (512KB)	882	4244	2763	1116	248
BlobMessage (2MB)	312	616	932	794	68
BlobMessage (4MB)	185	280	544	469	33

Tabulka 3: Výsledky výkonnostních testů pro testovací případ Q/NP/T

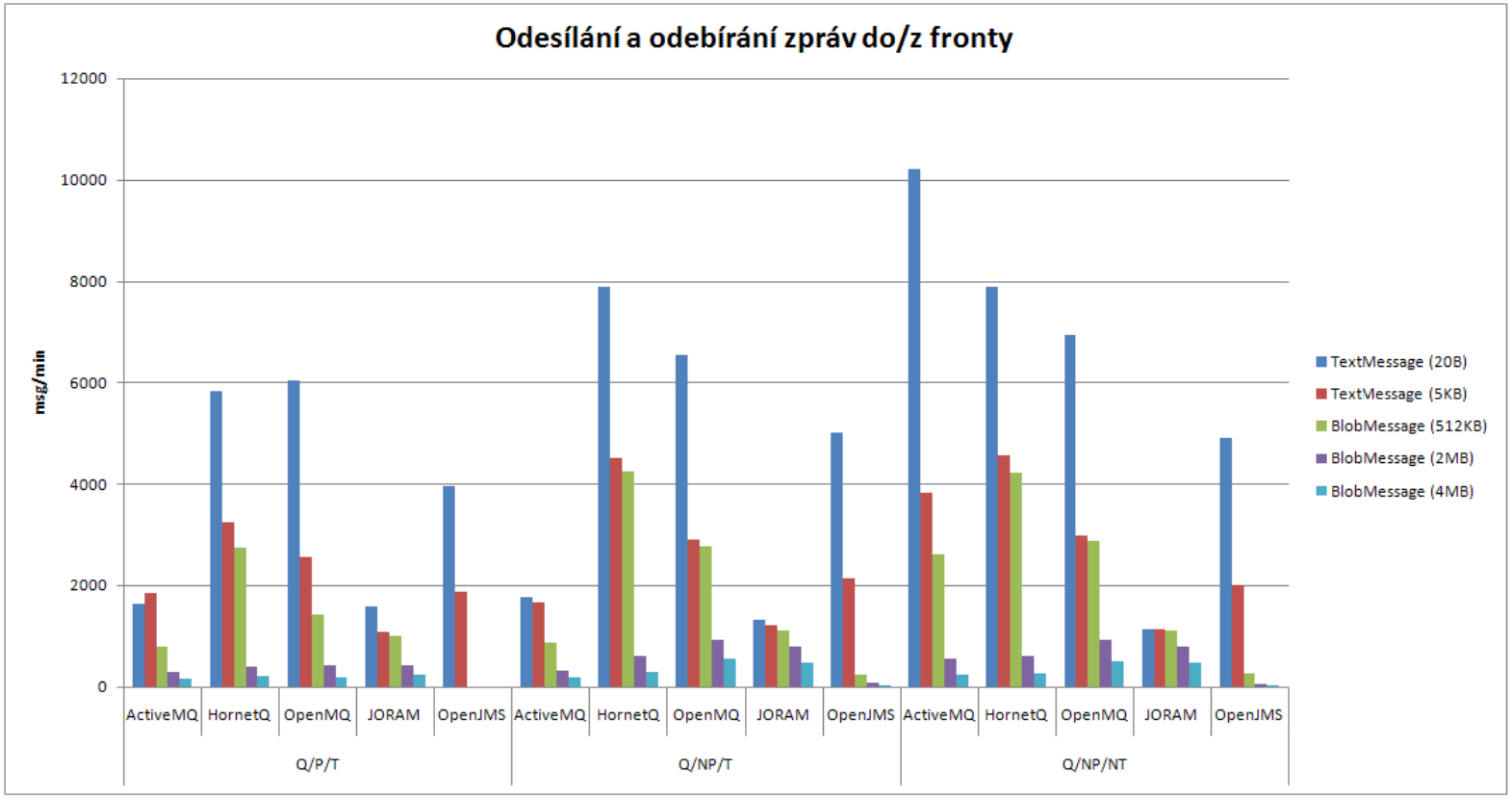
Poznámka: U JORAM messagingu docházelo k *celkovému pádu JMS severu pro velká přenášená data* a to především pro nepersistentní zprávy a netransakciovaný přenos. Toto selhávání se opakovalo i pro opakované testy. Viz tabulky. Příčinou může být to, že JORAM je *velice náročný na paměť*. Paměť pro JORAM server sice byla navýšena na maximální možnou hodnotu, ale i přesto asi byla nedostatečná.

Poznámka: U HornetQ docházelo (viz ** v tabulkách) k pádu připojení (cca po 20-25s). HornetQ je *velice citlivý na neukončená připojení* a neumožňuje nahradit staré Connection novým při jeho pádu, dokud původní nevyexpiruje na serveru. Tímto došlo k zablokování posílání i přijímání zpráv až do ukončení testu. Výsledky tak mohly být ještě vyšší.

Poznámka: U OpenJMS docházelo často k pádu jedné nebo více aplikací na straně klienta pro 20B zprávy a znemožněno navázání nového připojení. Tím nedošlo k zapsání výsledku do logu (viz * v tabulkách). Výsledná hodnota byla tedy stanovena na základě hodnot, které se podařilo získat a zbytek byl stanoven odhadem.

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	10217	7882**	6933	1130	4900*
TextMessage (5KB)	3814	4574	2980	1125	1998
BlobMessage (512KB)	2614	4232	2865	1110	275
BlobMessage (2MB)	552	617	918	787	64
BlobMessage (4MB)	243	268	495	482	35

Tabulka 4: Výsledky výkonnostních testů pro testovací případ Q/NP/NT



Obrázek 12: Graf s výsledky výkonnostních testů pro odesílání/odebírání zpráv do/z fronty

9.3.3 Výkonnostní testy zaměřené na odesílání zpráv

Testy spadající do této kategorie jsou podobné testům, které jsou popsány v předchozí kapitole. Opět zde jsou testovací případy tvořeny kombinací *persistence*, *různé velikosti zpráv*, *typem destinace* (fronta, topik) a *typem posílaných zpráv* (TextMessage nebo BlobMessage). Hlavní rozdíl je zde ten, že zde dochází *pouze k odesílání zpráv na JMS server*, ale nedochází k jejich konzumaci klienty.

Hlavním zaměřením tedy je zjištění výkonnosti jednotlivých serverů v případě, že dochází *pouze k odesílání zpráv*, jelikož *přijímání zpráv* částečně ovlivňuje výsledky zaměřené například na *persistenci*, protože v případě přijímání zpráv nehraje persistence významnou roli. Proto budou provedeny testovací případy s odesíláním persistentních a nepersistentních zpráv. Dále je zde například zajímavé porovnání s výsledky, kdy dochází i ke konzumaci zpráv, jak to bylo realizováno v předchozí kategorii testů. Další zajímá věc, která bude v těchto testech ověřována, vychází z *principu topiku*. Zprávy posílané do topiku jsou doručeny jednomu nebo více uživatelům, kteří jsou v aktuální chvíli (kdy dochází k odeslání zprávy) k topiku připojeni. Nepřipojení uživatelé zprávu již nikdy neobdrží. V případě, že není k topiku nikdo připojený, měl by server MS zprávy okamžitě zahazovat. To znamená, že při posílání zpráv do topiku bez jediného připojeného uživatele by mělo docházet k *maximální možné rychlosti posílání zpráv na server*. Pro tento typ testu jsem se inspiroval zde [41].

Testy budou realizovány již pouze pro způsob potvrzování doručení zpráv pomocí *transakcí*. Dále budou testy prováděny na destinacích typu *fronty* a *topiku bez trvale připojených odběratelů*.

Test byl prováděn s *pěti uživateli* (5 JMS klientů). Každý testovací případ běžel *60 sekund* a byl proveden *2 krát* nezávisle na sobě. Výsledek byl stanoven jako průměr získaných hodnot. Metrika zde byla *počet přenesených zpráv.*

Celkem bylo navrženo 5 testovacích případů, kde každý je prováděn pro 5 různých velikosti přenášených dat. Celkem tedy 25 testovacích případů.

Tyto testovací případy jsou:

1. posílání zpráv do topiku, kde není připojen žáden odběratel – persistentní
2. posílání zpráv do topiku, kde není připojen žáden odběratel – nepersistentní
3. posílání zpráv do topiku s připojenými odběrateli – nepersistentní
4. posílání zpráv do fronty s nepřipojenými odběrateli – persistentní
5. posílání zpráv do fronty s nepřipojenými odběrateli – nepersistentní

Jelikož výsledky těchto testů jsou docela rozsáhlé, některé tabulky a grafy byly zařazeny do příloh. Na ukázku zde byly zařazeny tabulky 5, 6 týkající se testovacích případů s frontou a k nim příslušející graf na obrázku 13. Tabulky 18, 19, 20 a graf na obrázku 20, týkajících se topiku, jsou zařazeny v přílohách.

Z těchto tabulek a grafů je možné nejlépe vypozorovat výsledky provedených testů této kategorie. Nyní zde zmíním některé nejdůležitější.

Některé plynoucí závěry z výsledků těchto testů:

- Nejlepších výsledků pro *odesílání zpráv do fronty v persistentním režimu* dosahovali HornetQ a OpenMQ. Pro větší velikosti dat i ActiveMQ. Jednoznačně nejhorší se zde jevil JORAM. V případě *zpráv do topiku* pak opět nejlepší HornetQ a OpenMQ. Ale ActiveMQ zde již ztrácí. Dále se zde opět ukazuje, že OpenJMS je pro velká data zcela nepoužitelný.
- U HornetQ a OpenMQ je vidět výrazný nárůst v počtu přenesených zpráv do fronty v nepersistentním režimu. V případě JORAM a OpenJMS je nárůst nižší. U ActiveMQ prakticky neexistuje žáden rozdíl mezi persistentním a nepersistentním režimem.
- Zajímavé, ale i očekávané, je, že u především OpenMQ dochází k především u velkých dat k výraznějšímu rozdílu mezi persistentním a nepersistentním přístupem.
- Zajímavé je porovnání těchto výsledků odesílání do front s výsledky z předchozí kategorie, kde jsou zprávy i konzumovány. Především v případě OpenMQ je patrné, že bude asi *nejrychlejší v přijímání zpráv*. Podobně také i např. u JORAM, HornetQ je vidět skoro *dvojnásobně více přenesených zpráv v případě*, že docházelo zároveň i ke konzumování zpráv. To ukazuje, že *skoro všechny odeslané zprávy byly i přijaty za dobu testu*. Jiná situace je ale u ActiveMQ. Zde jsou hodnoty, kdy docházelo pouze z odesílání zpráv, *vyšší* než v případě, kdy byly zároveň i přijímány, což ukazuje pravděpodobně na velké zatížení serveru, který pak nestíhal. Toto je asi také vysvětlení neočekávaně špatných výsledků ActiveMQ v prováděných testech. Toto mohlo být způsobeno nedostatečně výkonnou počítačovou sestavou.
- Totéž co v předchozím bodě je vidět i v testech prováděných na topiku, kde pro posílání zpráv má ActiveMQ slušné výsledky, ale v případě, že zároveň dochází ke konzumování, jsou výsledky dosti slabé. U ostatních MS je zde vidět několikanásobný rozdíl, což ukazuje na to, že tyto servery dobře zvládaly vytvořené zatížení.
- Podle očekávání se výsledky pro posílání persistentních a nepersistentních zpráv do netrvalého topiku *rovnají*, což odpovídá správné realizaci principu topiku, s částečnou výjimkou u OpenJMS.
- Pomocí netrvalého topiku byla stanovena maximální možná rychlost přenášení dat na server. Ukazuje se, že pro malá data nejrychleji přenáší data na server HornetQ, pro veliká data pak OpenMQ a JORAM.

Poznámka: Stejně jako u testů v předchozí kapitole u JORAM messagingu docházelo k celkovému selhání JMS severu pro velká přenášená data (viz *s.* v tabulkách). Taktéž jako v předchozím typu testů.

Poznámka: Taktéž jako v předchozí kapitole. U HornetQ docházelo (viz ** v tabulkách) pádu připojení (cca po 30 sekundách).

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	1971	3567	2916	628	2594
TextMessage (5KB)	1733	1847	1525	534	1299
BlobMessage (512KB)	730	890	876	472	-
BlobMessage (2MB)	323	382	202	s. (cca 50s)	-
BlobMessage (4MB)	189	209	101	s. (cca 25s)	-

Tabulka 5: Výsledky testů pro posílání zpráv do fronty s nepřipojenými odběrateli – persistentní

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	2083	3946**	2927	667	2615*
TextMessage (5KB)	1748	2224	1542	556	1433
BlobMessage (512KB)	755	1886	1807	488	273
BlobMessage (2MB)	323	455	486	s. (cca 50s)	61
BlobMessage (4MB)	185	256	244	s. (cca 25s)	34

Tabulka 6: Výsledky testů pro posílání zpráv do fronty s nepřipojenými odběrateli – nepersistentní

Poznámka: U OpenJMS docházelo často k pádu jedné nebo více aplikací na straně klienta pro 20B zprávy a znemožněno navázání nového připojení. Tím nedošlo k zapsání výsledku do logu (viz * v tabulkách). Viz totéž jako předchozí kapitole.

9.3.4 Výkonnostní testy na serverovou a topikovou škálovatelnost

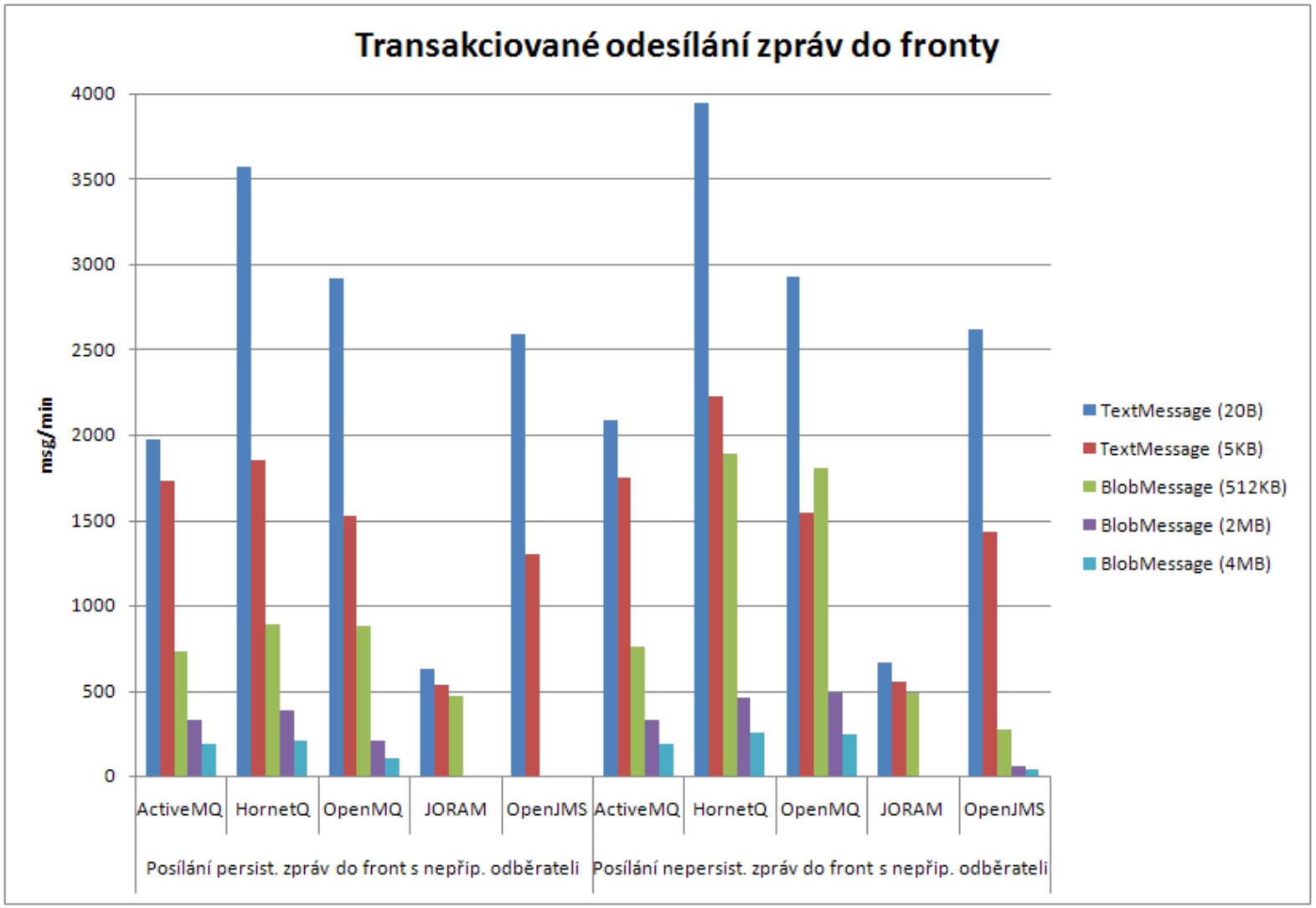
Následující testy jsou zaměřeny na *serverovou a topikovou škálovatelnost* jednotlivých MS. Inspiraci pro tyto testy jsem čerpal především zde [30]. Tyto testy jsou velmi vhodné pro zjištění výkonnosti serveru pro zvyšující se počet uživatelů, případně destinací.

„*Škálovatelnost nebo také rozšiřitelnost je žádoucí vlastnost systému nebo procesu, která má schopnost pracovat s neočekávanými změnami potřeby obsluhy čili zvyšovat sledované parametry v případě, že nastane taková potřeba*“ [42].

Serverová škálovatelnost pak zde znamená schopnost zvyšovat celkové množství přenesených zpráv při zvyšujícím se počtu uživatelů (klientů), kteří pracují na zvětšujícím se počtu topiků [30].

Topiková škálovatelnost znamená schopnost serveru zvyšovat celkové množství přenesených zpráv při zvyšujícím se počtu uživatelů na pevně daném počtu topiků (zde typicky na jednom) [30].

Při návrhu testovacích případů jsem vycházel ze dvou hlavních modelů použití, jak jsou zmíněny také například zde [30]. První je *nepersistentní zprávy do netrvalých topiků* (Non-Persistent Publishers & Non-Durable Subscribers). Tento model je většinou používán, pokud nám nezáleží na tom, zda zpráva bude doručena nebo nikoli, ale chceme dosáhnout co nejvyšší propustnosti. Druhým modelem je *persistentní zprávy do trvalých topiků* (Persistent Publishers & Durable Subscribers). Používán, pokud nám záleží na jistotě



Obrázek 13: Graf s výsledky výkonnostních testů pro odesílání zpráv do fronty

doručení zprávy i za cenu nižších rychlostí. Na základě těchto modelů pak byly navrženy 4 testovací případy.

Tyto testovací případy jsou:

1. Serverová škálovatelnost – persistentní zprávy, trvalý topik
2. Serverová škálovatelnost – nepersistentní zprávy, netrvalý topik
3. Topiková škálovatelnost – persistentní zprávy, trvalý topik
4. Topiková škálovatelnost – nepersistentní zprávy, netrvalý topik

Každý testovací případ serverové škálovatelnosti byl pak proveden pro *1/1/1* (počet odesílatelů/odběratelů/topiků), *5/5/5* a *15/15/15*. U topikové škálovatelnosti pak *1/1/1*, *5/5/1* a *15/15/1*. Dále bylo zvoleno *transakční potvrzování* pro všechny testy a velikost přenášených zpráv byla *512KB*. Zprávy byly přenášeny jako BlobMessage.

Každý testovací případ byl proveden *3 krát* nezávisle na sobě. Výsledek byl stanoven jako průměr získaných hodnot. Metrika zde byla *počet přenesených zpráv.*

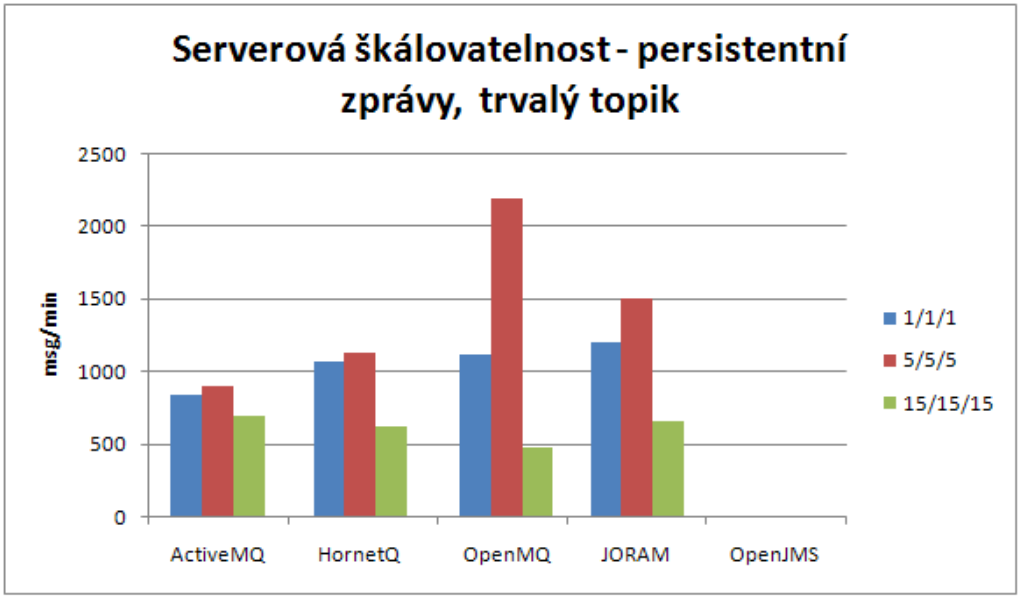
Nejlépe jsou výsledky těchto testů a z nich plynoucí závěry vidět v následujících tabulkách 7, 8, 9, 10 a grafech na obrázcích 14, 15, 16, 17.

Zhodnocení výsledků:

- Při testování se bohužel ukázalo, že na této počítačové sestavě slabšího výkonu, nebudou mít výsledky zrovna moc vypovídající hodnotu. Porovnání naměřených hodnot pro 1 a 5 odesílatelů/odběratelů je vcelku vypovídající. Horší je to s naměřenými hodnotami pro 15 odesílatelů/odběratelů, kdy často docházelo k „zasekání" celého počítače pro některé MS, hlavně pro OpenMQ, ale také pro JORAM nebo HornetQ, což jistě ovlivnilo výsledky, viz *** v tabulkách.
- Pro OpenJMS nebyly provedeny testovací případy s persistentními zprávami, jelikož OpenJMS umožňuje přenos persistentních zpráv jen do 32KB.
- Všechny testované MS vykazují nárůst počtu přenesených zpráv pro 5/5/5 u serverové a 5/5/1 u topikové škálovatelnosti. *Nejvyšší nárůst* je pak patrný u OpenMQ. Více viz tabulky a grafy.
- Pro 15/15/1 u topikové a 15/15/15 u serverové škálovatelnosti jsou již hodnoty výrazně nižší. Velký vliv na tyto výsledky bude mít pravděpodobně i nedostatečně výkonná počítačová sestava. A to především u OpenMQ nebo JORAM.
- Pro potvrzení výše uvedeného tvrzení, že tyto výsledky jsou ovlivněny slabou HW výkonností počítačové sestavy, na které pobíhaly testy, byl proveden testovací případ 15/15/1 pro OpenMQ na *výkonném školním serveru*. V zde naměřených výsledcích můžeme vidět, že pro 15/15/1 je OpenMQ JMS server přenese až 10x méně zpráv než pro 5/5/1. U experimentu provedeného na školní sestavě to bylo pouze cca 2,5x méně. Z těchto výsledků můžeme usuzovat, že OpenMQ sice poskytuje pro vyšší úroveň škálování horší výsledky, ale hlavním důvodem takto nízkých naměřených hodnot byla opravdu nedostatečná výkonnost testovací počítačové sestavy.

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
1/1/1	843	1065	1112	1198	-
5/5/5	904	1125	2184	1498	-
15/15/15	699***	629***	480***	662***	-

Tabulka 7: Výsledky testů pro serverovou škálovatelnost – persistentní zprávy, trvalý topik



Obrázek 14: Graf pro serverovou škálovatelnost – persistentní zprávy, trvalý topik

9.3.5 Testování doby pro připojení na server

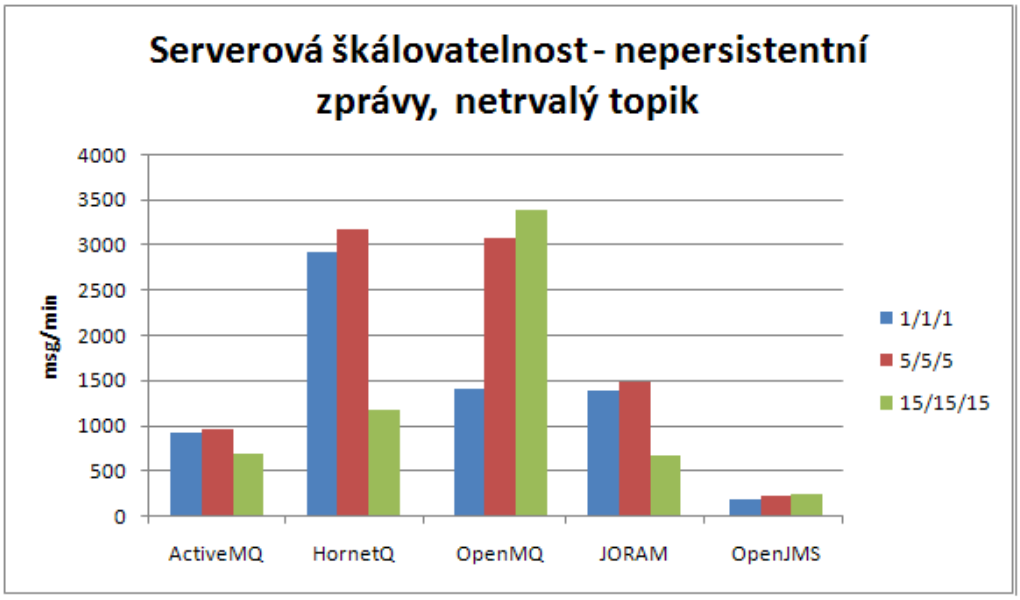
Doba potřebná k tomu, aby došlo k navázání spojení klienta na JMS server, může velice významně ovlivňovat celkovou výkonnost, především v případě, že dochází k vytváření nového připojení velice často.

Tento nejjednodušší test je zaměřený na změření této doby nutné na vytvoření připojení, to je od začátku vytváření ConnectionFactory až po zavolání metody start() na instanci Connection. Testování proběhlo celkem *3 krát*. Z naměřených hodnot byly vypočteny průměrné hodnoty uvedené v tabulce 11.

Z výsledků vyplývá, že *JORAM*, *OpenMQ* a *ActiveMQ* potřebují víceméně stejnou dobu na vytvoření připojení. Přibližně dvojnásobnou dobu pak potřebuju *OpenJMS*. Nejdéle pak trvalo připojit se klientům na server *HornetQ*.

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
1/1/1	927	2925	1402	1382	200
5/5/5	958	3166	3079	1489	240
15/15/15	687***	1170***	3383***	672***	248***

Tabulka 8: Výsledky testů pro serverovou škálovatelnost – nepersistentní zprávy, netrvalý topik



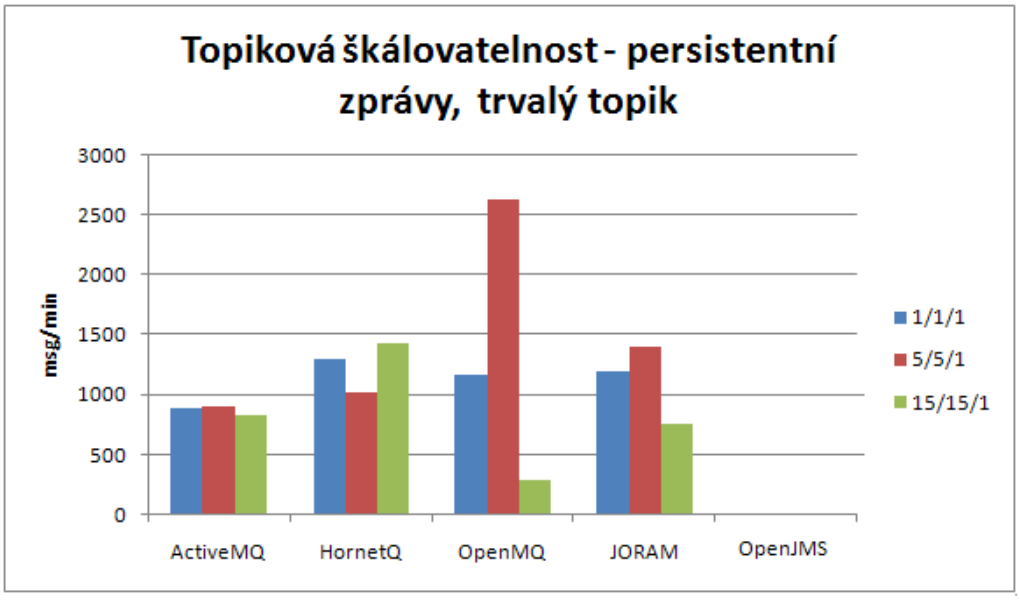
Obrázek 15: Graf pro serverovou škálovatelnost – nepersistentní zprávy, netrvalý topik

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
1/1/1	894	1301	1159	1194	-
5/5/1	908	1019	2622	1394	-
15/15/1	829***	1425***	295***	762***	-

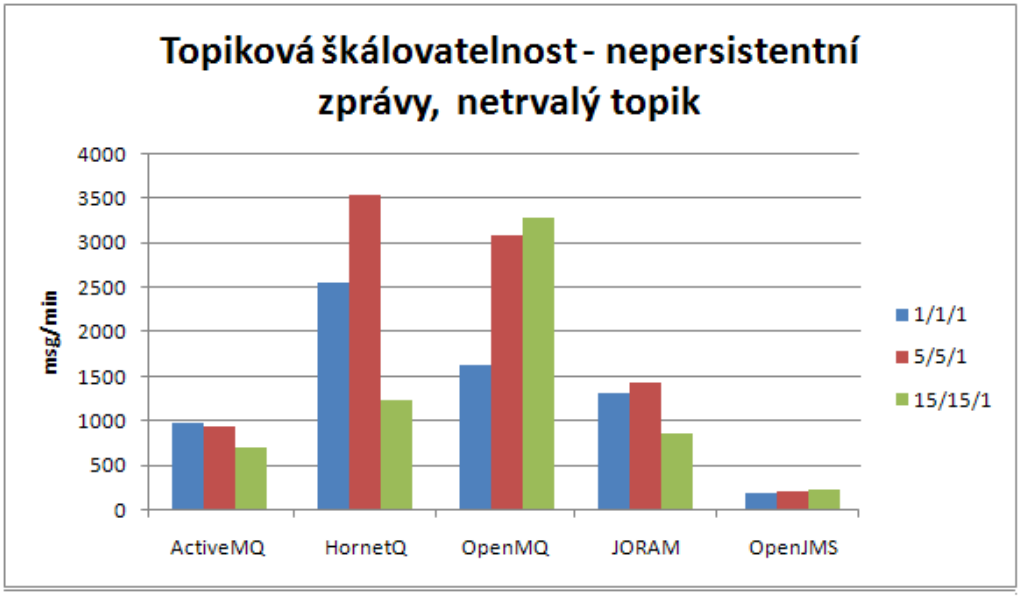
Tabulka 9: Výsledky testů pro topikovou škálovatelnost – persistentní zprávy, trvalý topik

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
1/1/1	976	2550	1634	1315	201
5/5/1	933	3531	3086	1424	224
15/15/1	701***	1234***	3279***	871***	235***

Tabulka 10: Výsledky testů pro topikovou škálovatelnost – nepersistentní zprávy, netrvalý topik



Obrázek 16: Graf pro topikovou škálovatelnost – persistentní zprávy, trvalý topik



Obrázek 17: Graf pro topikovou škálovatelnost – nepersistentní zprávy, netrvalý topik

	ActiveMQ	**HornetQ**	**OpenMQ**	**JORAM**	**OpenJMS**
čas pro navázání připojení [ms]	172,3	458,3	171,7	161,3	322,7

Tabulka 11: Doba nutná k vytvoření připojení na JMS server

10 Celkové zhodnocení

V této kapitole budou celkově zhodnoceny výsledky analýzy a porovnání testovaných MS. Budou zde shrnuty výsledky, které byly detailně popsány výše a to především z hlediska *výkonnostního*, ale i z hlediska *stability*, *jednoduchosti integrace s jinými softwarovými systémy* (především z hlediska obtížnosti nasazení MS), *přenositelnosti*, *udržovatelnosti* (dokumentace, pravidelné nové verze, atd.), *uživatelské přívětivosti*, *aj.*

ActiveMQ – tento MS patří mezi *nejkomplexnější* open source řešení (přidané vlastnosti, podpora dalších jazyků, atd.), viz kapitola 4.1.1. Velkou výhodou je také, že je asi vůbec n*ejoblíbenější* a *nejčastěji využívanou* nekomerční implementací. Podle mého názoru má velice kvalitně propracovanou a přehlednou dokumentaci. Na oficiálních stránkách je k dispozici „živé“ uživatelské fórum, kde není problém dostat odpověď na případné dotazy. Z mého pohledu bylo ActiveMQ také určitě *nejsnáze nasaditelné* a *nejvíce uživatelsky přívětivé* (webová konzole, dynamická tvorba destinací, přímé připojení na sever bez JNDI, atd.) viz kapitola 8.1. V pravidelných intervalech jsou vydávány nové stabilní verze a je zajištěna jejich stálá podpora. Při prováděných experimentech se ukázalo, že ActiveMQ je z vybraných implementací asi *nejstabilnější řešení* (nedocházelo prakticky vůbec k pádu serveru). Z výkonnostního hlediska se očekávalo, že bude v testech patřit k nejvýkonnějším. Při pohledu na výsledky to tak na první pohled nevypadá (částečně je to však způsobeno počítačovou sestavou, na které byly testy prováděny). Především pro *transakciovaný přenos zpráv* byly výsledky velmi slabé. Ale na druhou stranu pokud byl použit *netransakciovaný přenos*, patřilo ActiveMQ k jednomu z *nejlepších*, hlavně pak pro malá data. Při pohledu na existující porovnání (viz kapitola 9.1) se dá říct, že výsledky testů vcelku odpovídají výsledkům v těchto porovnáních.

HornetQ – také velice *komplexní* a *rychle se rozvíjející řešení* (viz kapitola 4.1.2). Je úzce spojen s ostatními projekty od společnosti JBoss, což je na jednu stranu výhoda (např. je integrován v JBoss Application Server) na druhou nevýhoda (např. horší orientace). Dokumentace je rozsáhlá ale vcelku přehledná. Taktéž jako u ActiveMQ je velkou výhodou „živé“ uživatelské fórum. Z mého pohledu je ale práce s HornetQ *náročnější* a *mnohem méně intuitivní* než je tomu u ActiveMQ. Příkladem může být například, že HornetQ neumožňuje dynamicky vytvářet destinace nebo „krkolomné“ zprovoznění administrační konzole (viz kapitola 8.2). Z hlediska udržovatelnosti jsou vydávány pravidelně nové verze a opravy nalezených vad, atd. Z výkonnostního hlediska se očekávalo, že HornetQ bude dosahovat pravděpodobně *nejlepších výsledků*. Toto se v provedených experimentech víceméně potvrdilo. Především pro malá data byl HornetQ jedním z *nejlepších MS* (viz kapitola 9). V testech se ukázalo je HornetQ je vcelku stabilní, jen pro malá data došlo někdy k pádu připojení, které již nešlo obnovit.

OpenMQ – již ne tak komplexní řešení jako předchozí dvě (viz kapitola 4.1.3). OpenMQ je vcelku *uživatelsky přívětivé* – snadné nasazení, přehledná administrační GUI aplikace, dynamicky vytvářené destinace, atd. Na druhou stranu veškerá dokumentace i oficiální stránky byly, podle mého názoru, *velmi nepřehledné* a nedalo se v nich dohledat potřebné informace (viz také kapitola 8.3). OpenMQ je také standardní součástí aplikační serveru GlassFish a může být integrován i s dalším systémy (Mule, atd.). Jedná se sice o pomaleji se rozvíjející, ale *trvale udržovaný projekt*, u kterého jsou pravidelně vydávány opravy chyb

i nové verze. Výsledky OpenMQ ve výkonnostních testech byly velkým překvapením. Podle existujících porovnání moc nenasvědčovalo tomu, že by OpenMQ mělo dosahovat nejlepších výsledků. Přesto ve většině provedených testů bylo *nejvýkonnějším* nebo jedním z *nejvýkonnějších MS*. Oproti ActiveMQ a HornetQ dosahovalo i velmi dobrých výsledků pro přenos dat o *velmi velkých velikostech*. Dobré výsledky také dosahovalo při *testech škálovatelnosti*. Při testech se ukázalo, že OpenMQ je dost *náročné na paměť*. Ta byla navýšena, ale i tak byla při některých testech nedostačující. I přes tyto problémy nedocházelo k pádu serveru, pouze zamezení posílání dalších zpráv odesílateli. Z tohoto důvodu se OpenMQ jeví jako *stabilní* (viz kapitola 9).

JORAM – již ne tak známy MS a také ne tak komplexní jako ActiveMQ nebo HornetQ (viz kapitola 4.1.4). Podobně jako OpenMQ se jedná o pomaleji se rozvíjející, ale *udržované řešení*, u kterého jsou pravidelně vydány nové verze. Integrace a nasazování JORAM JMS serveru patří k těm *obtížnějším* a *méně uživatelsky přívětivým*. JORAM totiž *neumožňuje dynamické vytváření destinací* a *defaultně nepodporuje persistentní zprávy nebo možnost přímého připojení k serveru.* Taktéž pro správu serveru a destinací nebyla k dispozici funkční uživatelsky přívětivá administrační konzole. Na druhou stranu je poskytováno propracované administrační API (viz kapitola 8.4). Jelikož tento MS není tak hojně využíván, je také problematické dohledat některé potřebné informace. Z hlediska provedených *výkonnostních testů* se jeví JORAM *velice rozdílně*. Pro *malé textové zprávy* JORAM dosahuje *hodně špatných výsledků*, dokonce někdy horších než OpenJMS. Naopak pro *velké zprávy* patří společně s OpenMQ k těm vůbec *nejlepším!* Také se ukázalo, že JORAM pravděpodobně umožňuje rychlejší přenos zpráv ze serveru (konzumování) než na server (posílání). Z hlediska stability JORAM *nebyl příliš stabilní* a často docházelo k celkovému pádu serveru (u jiných MS docházelo většinou pouze k pádu připojení na straně klienta či zamezení posílání dalších zpráv na server). Toto bylo pravděpodobně způsobeno velkými nároky na paměť, která už ale nemohla být ale více navýšena (více viz kapitola 9).

OpenJMS – *nejjednodušší* a *nejméně propracovaný MS*, který již několik let nevyvíjí. Přestože již není udržovaný, je poměrně známý. Vzhledem k tomu, že již není vyvíjen a malému množství poskytovaných vlastností se určitě nehodí pro použití v některé reálné aplikaci. Na druhou stranu pro svoji jednoduchost je vhodný při seznamování se s MOM a JMS. Nasazování a práce s OpenJMS byla *docela snadná* vzhledem k jeho jednoduchosti a *přehledné dokumentaci* (viz kapitola 4.1.7 a 8.5). Výsledky výkonnostního testování vytváří zajímavý kontrast mezi produktem, jehož vývoje se před několika lety zastavil a dnešními nejrozšířenějšími a nejvýkonnějšími otevřenými implementacemi. Výsledky testů jednoznačně ukazují, že OpenJMS je, ve většině případů, *několikanásobně „pomalejší "* než ostatní MS. Ale překvapením je, že pro *malé textové zprávy* dokáže poskytovat velice *slušné výsledky*, i lepší než jiné MS. Během testů se ukázalo, že OpenJMS je spíše *méně stabilní*. Především pro malá data pravidelně docházelo k selhání na straně některých klientů, ale na druhou stranu nedocházelo k pádu celého serveru (viz kapitola 9).

Všechny analyzované messaging systémy jsou postaveny na Javě, tudíž jsou *velmi dobře přenositelné.*

11 Závěr

Cílem této diplomové práce bylo analyzovat nejrozšířenější stávající otevřené messaging systémy, popsat je, a v praktických experimentech ověřit jejich výkonnost pro přenos velkého množství zpráv. Během tvorby této práce jsem se seznamoval se základními principy a vlastnostmi messaging systémů (MOM) založených na asynchronní komunikaci a jejich významem v oblasti systémové integrace. Toto jsem také popsal v první části práce. V další části jsem pak věnoval nejznámějším komerčním a otevřeným implementacím MOM a uvedl jejich použití v dalších systémech.

Pro samotné výkonnostní testování jsem navrhl a naimplementoval testovací aplikaci, pro kterou byla nalezena motivace v kombinaci aplikací pro sociální sítě, aplikací typu instant messagingu a emailové komunikace. Tato konzolová aplikace implementovaná v Javě byla navržena jako tlustý klient, kde vzájemná interakce mezi jednotlivými klienty probíhá právě skrze MOM, který tvoří komunikační páteř této distribuované aplikace. Jednotlivé funkce aplikace byly navrženy tak, aby mohly být co možná nejlépe otestovány základní vlastnosti MOM u jednotlivých vybraných implementací messaging systémů. Kromě testovací aplikace byl také naimplementován adaptér (JMS Adaptér), který umožňuje aplikaci skoro jednotným způsob pracovat s vybranými implementacemi. Celkem bylo vybráno a nasazeno 5 implementací MOM. A to ActiveMQ, HornetQ, OpenMQ, JORAM a OpenJMS.

Následně byly navrženy a provedeny výkonnostní experimenty s těmito vybranými nasazenými implementacemi. Navržené testy byly rozděleny do 5 kategorií – testy na celé aplikaci (použity všechny realizované funkce), testy vybraných vlastností (pro různé konkrétní situace použití), testy zaměřené na odesílání zpráv, testy na serverovou a topikovou škálovatelnost a testování doby pro vytvoření připojení na server. Popis, výsledky ve formě tabulek a grafů a vyhodnocení těchto testů tvoří největší část této práce.

Nakonec bylo provedeno celkové zhodnocení výsledků jednotlivých messaging systémů a to především z výkonnostního hlediska, ale i z hlediska stability, jednoduchosti integrace s jinými SW systémy, udržovatelnosti, přenositelnosti, uživatelské přívětivosti a dalších.

Co se týká dalšího rozvoje této práce, bylo by vhodné se zaměřit na další otevřené implementace, které zde nebyly testovány. Případně provést testy na výkonnější počítačové sestavě.

12 Reference

[1] KHUDHUR, Patrik a Lukáš ERBEN. *Propojování roztříštěné infrastruktury* [online]. 2008. [cit. 2012-03-28]. Dostupné z: http://businessworld.cz/soa-a-eai/propojovani-roztristene-infrastruktury-1786.

[2] PÁLOS, Gustáv. *Komunikácia aplikácií v informačnom systéme Univerzity Komenského.* Bratislava, 2006. Diplomová práce. Univerzita Komenského, Fakulta matematiky, fyziky a informatiky. Vedoucí práce Mgr. Pavol Mederly.

[3] CURRY, Edward. *Message-Oriented Middleware* [online]. 2004. [cit. 2012-03-28]. Dostupné z: http://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf.

[4] HOHPE, Gregor a Bobby WOOLF. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2004. ISBN 0-321- 20068-3.

[5] ZAPLETAL, Lukáš. *Middleware orientovaný na zprávy* [online]. 2010. [cit. 2012-03-28]. Dostupné z: http://knol.google.com/k/luk%C3%A1%C5%A1-zapletal/middleware-orientovan%C3%BD-na-zpr%C3%A1vy/1as80wv4bdzca/7-#Middleware_orientovan%28C3%29%28BD%29_na_zpr%28C3%29%28A1%29vy-Message_oriented_middleware_%2828%29MOM%2829%29_je-_kategori%28C3%29%28AD%29_softwaru_zaji%28C5%29%28A1%29%28

[6] *Java Message Service (JMS)* [online]. [cit. 2012-03-28]. Dostupné z: http://www.oracle.com/technetwork/java/jms/index.html.

[7] ORASKARI, Jyrki. *The Performance of Open Message-Oriented Middleware Protocols in Smart Space Access.* Espoo, 2010. Diplomová práce. Aalto University, Faculty of Information and Natural Sciences. Dostupné z: http://www.scribd.com/doc/55822666/20/STOMP-Streaming-Text-Oriented-Message-Protocol.

[8] *Java^{TM} Message Service Tutorial* [online]. [cit. 2012-03-28]. Dostupné z: http://docs.oracle.com/javaee/1.3/jms/tutorial/index.html.

[9] ZAPLETAL, Lukáš. *Java Message Service 1.1* [online]. 2010. [cit. 2012-03-28]. Dostupné z: http://knol.google.com/k/luk%C3%A1%C5%A1-zapletal/java-message-service-1-1/1as80wv4bdzca/6#.

[10] *Novell exteNd Java Message Service 5.2 Tutorial* [online]. 2004. [cit. 2012-03-28]. Dostupné z: http://www.novell.com/documentation/extend52/Docs/help/MP/-jms/tutorial/.

[11] ERL, Thomas. *SOA Servisně orientovaná architektura: Kompletní průvodce.* Computer Press. 2009. 672 s. ISBN 978-80-251-1886-3.

[12] W3C. *Web Services Architecture* [online]. ©2002. [cit. 2012-03-28]. Dostupné z: http://www.w3.org/TR/ws-arch/.

[13] VACEK, Štěpán. Distribuované transakce. *Systémová integrace* [online]. 2010, roč. 17, č. 3, s. 25 [cit. 2012-03-28]. ISSN 1210-9479. Dostupné z: www.cssi.cz/cssi/system/files/all/si-2010-03-02-vacek.pdf.

[14] KOČÍ, Vladimír. *Integrácia aplikácií pomocou podnikovej zbernice služieb.* Bratislava, 2007. Diplomová práce. Univerzita Komenského, Fakulta matematiky, fyziky a informatiky. Vedoucí práce Mgr. Pavol Mederly.

[15] ANTOŠ, Jan. *Role ESB v servisně orientovaných architekturách* [online]. 2008. [cit. 2012-03-28]. Dostupné z: http://si.vse.cz/archive/proceedings/2008/role-esb-v-servisne-orientovanych-architekturach.pdf.

[16] SCHREINER, Vladimír. *Implementace SOA pomocí moderních ICT principů.* Brno, 2007. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Mgr. Jan Pavlovič. Dostupné z: http://is.muni.cz/th/60471/fi_m/dp.pdf.

[17] ŠTUMPF, Jindřich. *Podniková sběrnice služeb* [online]. 2006. [cit. 2012-03-28]. Dostupné z: http://www.galeos.cz/uploads/Soubory/ClankySOI/-Podnikova_sbernice_sluzeb.pdf.

[18] CHAPPELL, David. *Enterprise Service Bus: Theory in Practice.* Sebastopol: O'Reilly, 2004, 247 s. ISBN 978-0-596-00-675-4.

[19] *Apache ActiveMQ^{TM}* [online]. ©2004–2011. [cit. 2012-03-28]. Dostupné z: http://activemq.apache.org/

[20] Apache ActiveMQ. In: *Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-03-28]. Dostupné z: http://en.wikipedia.org/wiki/Activemq.

[21] *HornetQ - putting the buzz in messaging - JBoss Community* [online]. 2012. [cit. 2012-03-28]. Dostupné z: http://www.jboss.org/hornetq/.

[22] *HornetQ 2.0: rekordně rychlý messaging* [online]. 2010. [cit. 2012-03-28]. Dostupné z: http://www.jboss.cz/content/hornetq-20-rekordne-rychly-messaging.

[23] *Open Message Queue : Open Source Java Message Service (JMS) – Java.net* [online]. ©2008–2012. [cit. 2012-03-28]. Dostupné z: http://mq.java.net/.

[24] *JORAM: Java (TM) Open Reliable Asynchronous Messaging* [online]. ©1999–2010. [cit. 2012-03-28]. Dostupné z: http://joram.ow2.org/.

[25] JORAM. In: *Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001– [cit. 2012-03-28]. Dostupné z: http://en.wikipedia.org/wiki/JORAM.

[26] *Apache Qpid^{TM}: Open Source AMQP Messaging* [online]. ©2004–2011. [cit. 2012-03-28]. Dostupné z: http://qpid.apache.org/.

[27] Qpid. *Mule ESB* [online]. 2012. [cit. 2012-03-28]. Dostupné z: http://www.mulesoft.org/qpid.

[28] *RabbitMQ* [online]. ©2012. [cit. 2012-03-28]. Dostupné z: http://www.rabbitmq.com/.

[29] *OpenJMS* [online]. ©1999–2007. [cit. 2012-03-28]. Dostupné z: http://openjms.sourceforge.net/.

[30] *JMS Performance Comparison* [online]. 2011. [cit. 2012-03-28]. Dostupné z: http://www.fiorano.com/docs/jms_performance_comparison.pdf.

[31] *FioranoMQ Enterprise Messaging* [online]. ©2012. [cit. 2012-03-28]. Dostupné z: http://www.fiorano.com/products/Enterprise-Messaging/JMS/Java-Message-Service/FioranoMQ.php.

[32] MENTH, Michael, Robert HENJES, Christian ZEPFEL a Sebastian GEHRSITZ. *Throughput Performance of Popular JMS Servers* [online]. 2006. [cit. 2012-03-28]. Dostupné z: http://www.sciweavers.org/publications/throughput-performance-popular-jms-servers.

[33] *Progress SonicMQ for Enteprise Messaging* [online]. ©1993–2012. [cit. 2012-03-28]. Dostupné z: http://www.progress.com/en/sonic/sonicmq.html.

[34] SonicMQ. *GALEOS* [online]. ©2010-2012. [cit. 2012-03-28]. Dostupné z: http://www.galeos.cz/produkty/sonic/sonicmq.

[35] *IBM - WebSphere MQ - Software* [online]. [2012]. [cit. 2012-03-30]. Dostupné z: http://www-01.ibm.com/software/integration/wmq/.

[36] GAMBLIN, Richard. WEBSPHERE USER GROUP. *A Client Story: PCI Compliance with WebSphere MQ Advanced Message Security* [online]. 2011. [cit. 2012-03-30]. Dostupné z: http://www.websphereusergroup.org.uk/wug/-files/presentations/31/Richard_Gamblin_-_PCI_with_WMQ_AMS_&_FTE.pdf.

[37] LEŠTINA, Petr. *SOA - Enterprise Service Bus* [online]. 2008. [cit. 2012-03-30]. Dostupné z: http://www.common.cz/attachments/104_petr_lestina_ibm_integrace_esb.pdf.

[38] Twitter message queues move to Scala. In: *The Scala Programming Language* [online]. 2009. [cit. 2012-03-30]. Dostupné z: http://www.scala-lang.org/node/1008.

[39] Performance testing open source JMS part 1. In: *C2B2 - Laying the Foundations for Enterprise Scale Java* [online]. 2008. [cit. 2012-03-30]. Dostupné z: http://www.c2b2.co.uk/iPoint/ipoint?SelectedPage=69&110ArticleID=17.

[40] *JMS speed test: ActiveMQ vs HornetQ* [online]. 2011. [cit. 2012-03-30]. Dostupné z: http://integr8consulting.blogspot.com/2011/02/jms-speed-test-activemq-vs-hornetq.html.

[41] CHIRINO, Hiram. *STOMP Messaging Benchmarks: ActiveMQ vs Apollo vs HornetQ vs RabbitMQ* [online]. 2011. [cit. 2012-03-30]. Dostupné z: http://hiramchirino.com/blog/2011/12/stomp-messaging-benchmarks-activemq-vs-apollo-vs-hornetq-vs-rabbitmq/.

[42] Škálovatelnost. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-03-30]. Dostupné z: http://cs.wikipedia.org/wiki/%C5%A0k%C3%A1lovatelnost.

A Grafy a tabulky

Zde jsou umístěny tabulky a grafy, které zachycují výsledky výkonnostních testů. Jedná se o tabulky a grafy týkající se testů popsaných v kapitolách 9.3.1 a 9.3.2.

Poznámka: U JORAM messagingu docházelo k celkovému pádu (selhání) JMS severu pro velká přenášená data (viz *s.* v tabulkách) a to především pro nepersistentní zprávy a netransakciovaný přenos. Příčinou mohlo být to, že JORAM je velice náročný na paměť, která sice byla navýšena na maximální možnou hodnotu, ale i přesto asi byla nedostatečná.

Poznámka: * v tabulkách znamená, že u OpenJMS docházelo často k pádu jedné nebo více aplikací na straně klienta pro 20B zprávy a znemožněno navázání nového připojení, a tím nedošlo k zapsání výsledků. Výsledná hodnota byla tedy pak stanovena odhadem na základě hodnot, které se podařilo získat.

Poznámka: ** v tabulkách znamená, že došlo k pádu připojení (cca po 20-25 sekundách) u HornetQ. Jelikož je HornetQ velice citlivý na neukončená připojení došlo k zablokování posílání i přijímání zpráv až do ukončení testu.

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	1764	19494**	12733	3185	7384*
TextMessage (5KB)	1840	9650	6462	2433	2402
BlobMessage (512KB)	873	3018	3233	1550	-
BlobMessage (2MB)	300	698	986	612	-
BlobMessage (4MB)	174	280	498	s.(cca 35s)	-

Tabulka 12: Výsledky výkonnostních testů pro testovací případ T/P/T

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	1902	19144**	14001	3828	11888*
TextMessage (5KB)	1901	9204	6978	2880	2399
BlobMessage (512KB)	797	3009	3288	1600	236
BlobMessage (2MB)	300	529	963	628	65
BlobMessage (4MB)	169	268	499	s.(cca 35s)	34

Tabulka 13: Výsledky výkonnostních testů pro testovací případ T/NP/T

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	24757	19240**	16451	4235	12012*
TextMessage (5KB)	7424	9372	7108	3041	2296
BlobMessage (512KB)	3337	3662	3420	2407	253
BlobMessage (2MB)	482	601	995	s.(cca 55s)	64
BlobMessage (4MB)	243	287	522	s.(cca 25s)	35

Tabulka 14: Výsledky výkonnostních testů pro testovací případ T/NP/NT

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	1554	5905	13550	4078	3541*
TextMessage (5KB)	1683	4440	6899	3314	1757
BlobMessage (512KB)	768	1169	2343	1590	-
BlobMessage (2MB)	279	410	479	576	-
BlobMessage (4MB)	175	228	216	270	-

Tabulka 15: Výsledky výkonnostních testů pro testovací případ DT/P/T

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	2282	19684**	13838	3742	9881
TextMessage (5KB)	2290	8891	7053	2753	3153
BlobMessage (512KB)	874	3360	3266	1669	232
BlobMessage (2MB)	300	534	979	589	64
BlobMessage (4MB)	172	279	439	s.(cca 25s)	35

Tabulka 16: Výsledky výkonnostních testů pro testovací případ DT/NP/T

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	25021	19764**	16344	3030	11709*
TextMessage (5KB)	8456	9981	5927	2715	3307
BlobMessage (512KB)	3211	3153	3392	2234	260
BlobMessage (2MB)	530	542	842	s.(cca 45s)	65
BlobMessage (4MB)	244	266	476	s.(cca 25s)	35

Tabulka 17: Výsledky výkonnostních testů pro testovací případ DT/NP/NT

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	2094	3950**	2545	900	2924
TextMessage (5KB)	1927	2387	1494	571	1548
BlobMessage (512KB)	463	1381	2018	711	-
BlobMessage (2MB)	231	421	685	455	-
BlobMessage (4MB)	158	241	482	353	-

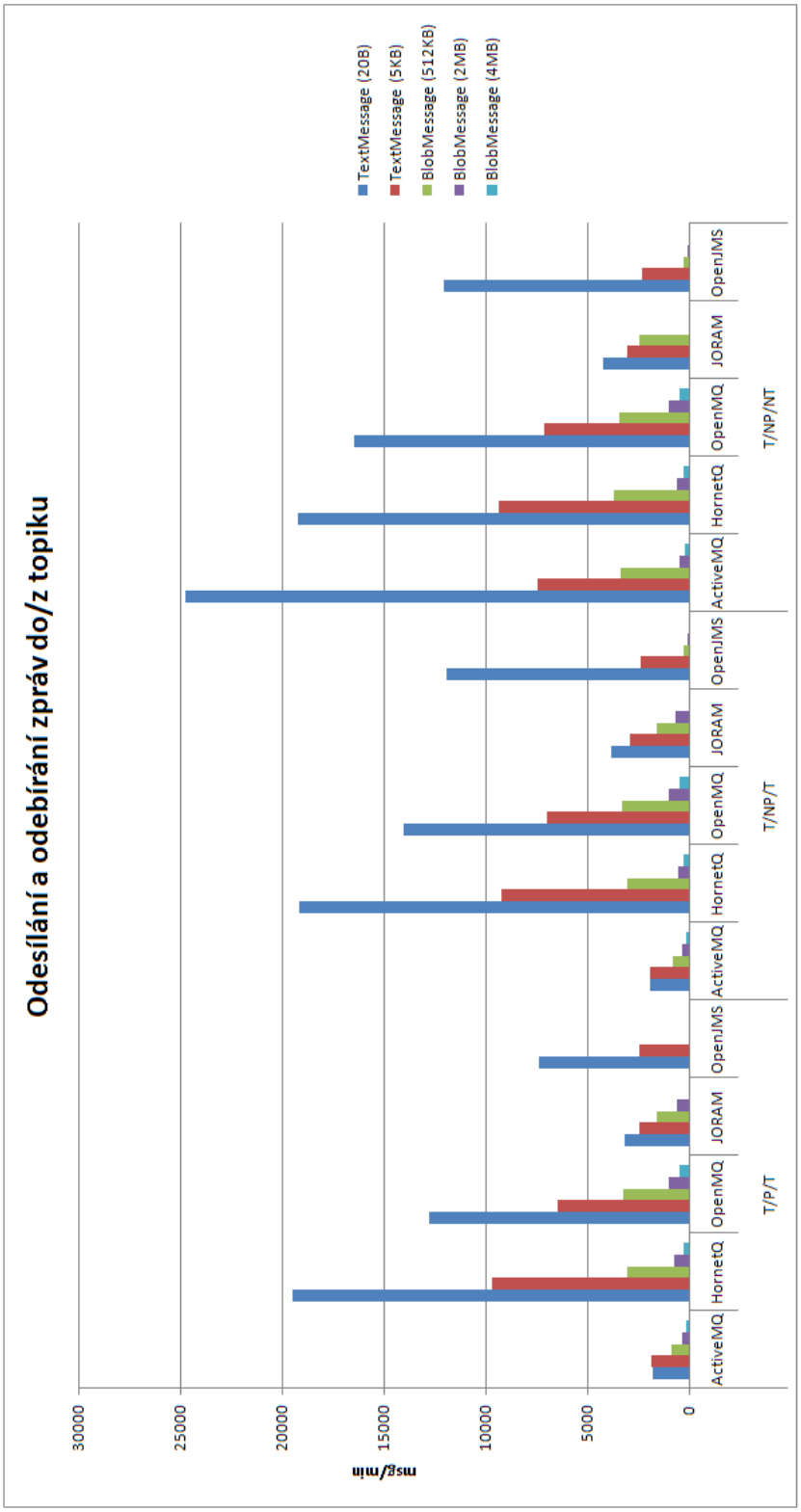
Tabulka 18: Výsledky testů pro posílání zpráv do topiku, kde není připojen žáden odběratel – persistentní

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	2249	3956**	2956	678	3450*
TextMessage (5KB)	1802	2398	1549	636	1722
BlobMessage (512KB)	684	1419	2055	589	268
BlobMessage (2MB)	318	485	722	472	66
BlobMessage (4MB)	178	260	462	362	34

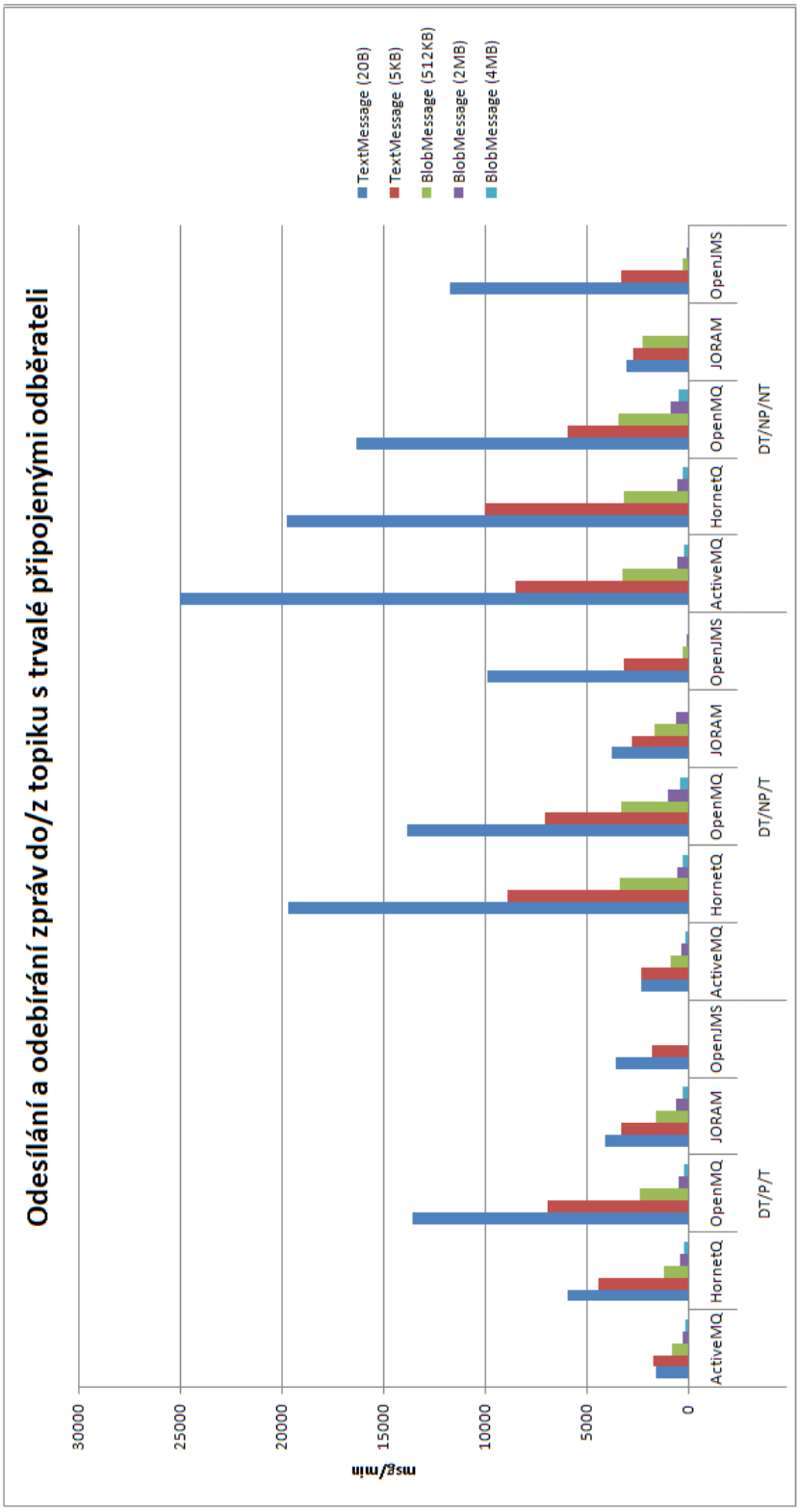
Tabulka 19: Výsledky testů pro posílání zpráv do topiku, kde není připojen žáden odběratel – nepersistentní

	ActiveMQ	HornetQ	OpenMQ	JORAM	OpenJMS
TextMessage (20B)	2149	19723**	12252	3026	10165
TextMessage (5KB)	2368	9165	6704	2673	2486
BlobMessage (512KB)	928	2806	3139	1463	233
BlobMessage (2MB)	277	932	940	567	63
BlobMessage (4MB)	179	313	502	s. (cca 35s)	33

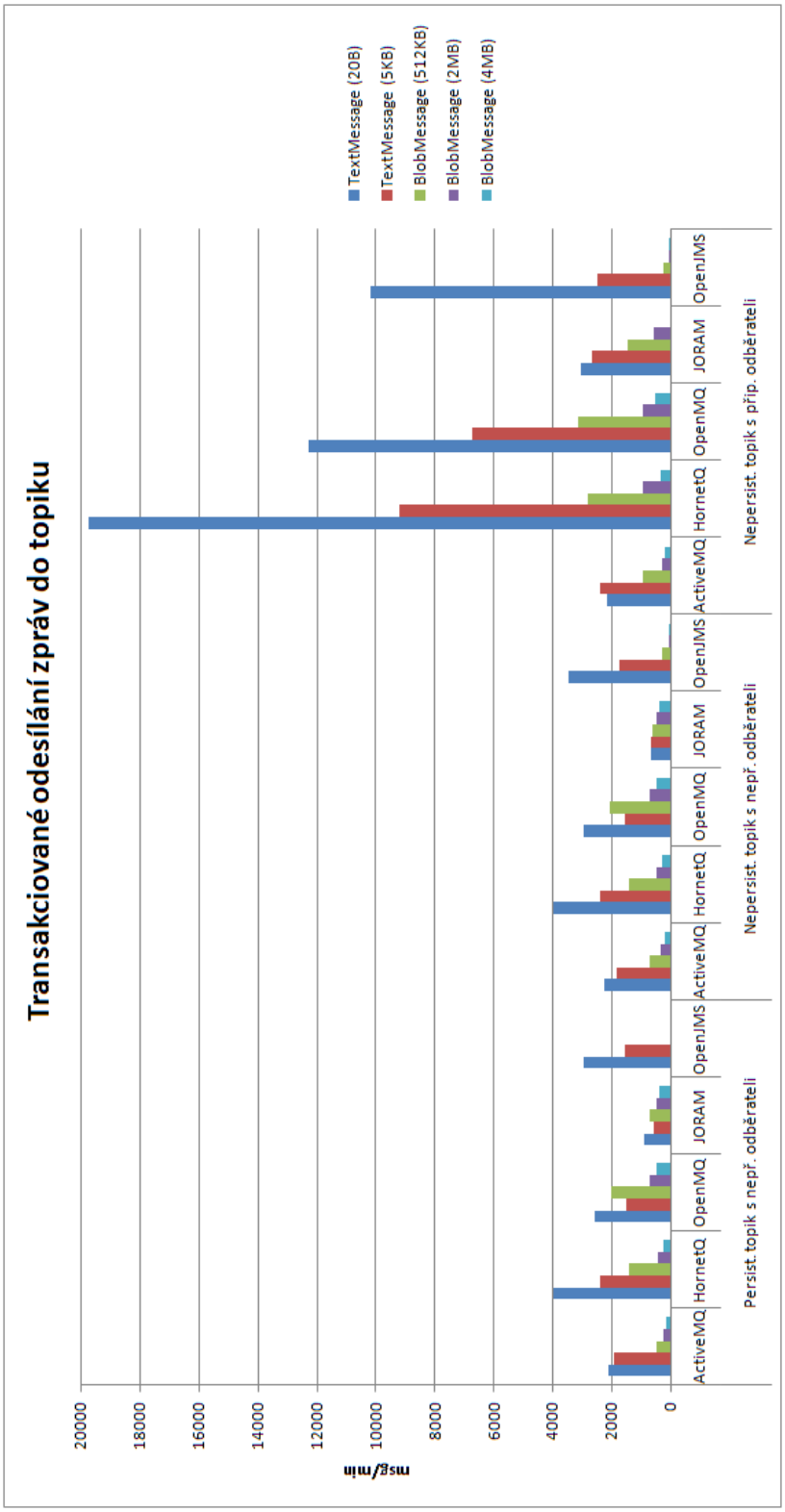
Tabulka 20: Výsledky testů pro posílání zpráv do topiku s připojenými odběrateli – nepersistentní



Obrázek 18: Graf s výsledky výkonnostních testů pro odesílání/odebírání zpráv do/z topiku



Obrázek 19: Graf s výsledky výkonnostních testů pro odesílání/odebírání zpráv do/z trvalého topiku



Obrázek 20: Graf s výsledky výkonnostních testů pro odesílání zpráv do topiku

B Obsah přiloženého CD

- Diplomová práce v elektronické podobě
- Testovací aplikace pro vybrané MS
- Upravená verze testovací aplikace pro účely testování
- JMS Adaptér
- Knihovny nasazovaných messaging systémů
- Testovací skripty používané pro zautomatizování provádění testů
- Testovací data (el. dokumenty), které byly přenášeny
- Logy s naměřenými hodnotami, na základě kterých byly pak sestaveny příslušné tabulky a grafy