

Nástroj pro zpracování výsledků měření v Red Hat Enterprise MRG

Measurement Control and Processing Tool
for Red Hat Enterprise MRG

Bc. Petra Svobodová



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Petra Svobodová**
Osobní číslo: **A11504**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **kombinovaná**

Téma práce: **Nástroj pro zpracování výsledků měření v Red Hat Enterprise MRG**

Zásady pro vypracování:

1. Vytvořte rešerši na téma Red Hat Enterprise MRG s důrazem kladeným na MRG Messaging.
2. Vytvořte nástroj pro zpracování výsledků měření objemu přijatých a odeslaných zpráv a časové odezvy Red Hat Enterprise MRG Messaging brokeru.
3. Pro každou kombinaci vstupních parametrů nástroje dle specifikace proveďte měření, výpočet souhrnných informací a export výsledků do souboru.
4. Vytvořte uživatelskou a programovou dokumentaci.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **TŮMOVÁ, Olga. Metrologie a hodnocení procesů. 1. vyd. Praha: BEN – technická literatura, 2009, 231 s. ISBN 978-80-7300-249-7.**
2. **ANDĚL, Jiří. Základy matematické statistiky. Vyd. 3. Praha: Matfyzpress, 2011, 358 s. ISBN 978-80-7378-162-0.**
3. **BUDÍKOVÁ, Marie, Tomáš LERCH a Štěpán MIKOLÁŠ. Základní statistické metody. 1. vyd. Brno: Masarykova univerzita, 2005, viii, 170 s. ISBN 80-210-3886-1.**
4. **Red Hat Enterprise MRG: Customer Portal [online]. 2013 [cit. 2013-01-18]. Dostupné z: https://access.redhat.com/knowledge/docs/Red_Hat_Enterprise_MRG/**
5. **PILGRIM, Mark. Ponořme se do Python(u) 3: Dive into Python 3. Praha: Cz.Nic, 2010, 430 s. CZ.NIC. ISBN 978-80-904248-2-1.**

Vedoucí diplomové práce:

Ing. Michal Bližňák, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

22. února 2013

Termín odevzdání diplomové práce:

22. května 2013

Ve Zlíně dne 22. února 2013

prof. Ing. Vladimír Vašek, CSc.

děkan

L.S.

doc. Mgr. Roman Jašek, Ph.D.

ředitel ústavu

ABSTRAKT

Cílem diplomové práce je návrh a implementace aplikace „qpid_benchmark_stats“ pro zpracování měření propustnosti a časové odezvy produktu Red Hat Enterprise MRG Messaging. Aplikace je vyvíjena pod licencí Apache 2.0 a bude sloužit testovacímu týmu společnosti Red Hat k přesnějšímu určení naměřených hodnot propustnosti (throughput) a časové odezvy (latency) Red Hat Enterprise MRG Messaging brokeru, získaných s využitím pomocných nástrojů „qpid-cpp-benchmark“, „qpid-send“ a „qpid-receive“. Pomocné nástroje jsou dostupné na webových stránkách Apache-SVN [1] a pro účely práce byly mírně upraveny. Sledovanými parametry produktu Red Hat Enterprise MRG Messaging jsou celková propustnost zpráv, propustnost odesílacího i přijímacího klienta a latence zpráv.

Aplikace „qpid_benchmark_stats“ nejprve načte vstupní parametry, ověří jejich formát, pak ve smyčce spouští „qpid-cpp-benchmark“ a provede potřebný počet měření. V naměřených hodnotách odhalí odlehlé hodnoty a měření, která tyto hodnoty obsahují, vyloučí. Pak spočítá průměr, směrodatnou odchylku průměru (střední kvadratickou chybu), minimální a maximální hodnotu pro každou sledovanou veličinu. Nakonec naměřené hodnoty i výsledky serializuje a uloží do souboru pro pozdější použití. Součástí balíčku se zdrojovými kódy je také aplikace „compare_data.py“, která demonstruje využití naměřených dat pro vzájemná porovnávání.

V teoretické části práce je popsán projekt Red Hat Enterprise MRG s důrazem na komponentu Messaging a použité statistické metody zpracování dat, praktická část obsahuje specifikaci požadavků, programovou a uživatelskou dokumentaci nového nástroje.

Klíčová slova:

Messaging, měření, zpracování dat, software, open source, latence, propustnost, statistika, odezva

ABSTRACT

The main goals of this diploma thesis are “qpid_benchmark_stats” application architecture and implementation for Red Hat Enterprise MRG Messaging system throughput and latency measurement including later data processing. The application is developed under the Apache license 2.0 and will be used by Red Hat MRG testing team to get more accurate message throughput and latency performance statistics, measured by support tools “qpid-cpp-benchmark”, “qpid-send” and “qpid-receive”. The support tools are available at Apache-SVN web page [1] and were slightly modified for purpose of this thesis. The most important performance parameters to measure are total message throughput, sender and receiver message throughput and finally message latency.

The “qpid_benchmark_stats” application reads the input options, parses them and checks their formats; then it runs “qpid-cpp-benchmark” and executes the required number of measurements; finds extremes and removes measurements which contain the extreme values. After that the application computes average value, standard deviation of the average value (mean square error), minimum and maximum value for every monitored variable. In the end all measured and computed values are serialized and saved into a file for later use. The source code package contains also an example application “compare_data.py” what shows how to use the archived data for mutual comparisons.

In the theoretical part the project Red Hat Enterprise MRG, with an emphasis on the Messaging part, and used statistical methods are described. The practical part contains the requirements specification, program and user documentation.

Keywords:

Messaging, measurement, data processing, software, open source, latency, performance, statistics

Ráda bych poděkovala panu ing. Michalovi Bližňákovi, Ph.D., vedoucímu práce, za konzultace a cenný čas, který mi věnoval, panu ing. Františkovi Řezníčkovi, mému odbornému konzultantovi, kolegovi a kamarádovi za jeho odbornou pomoc a za všechno, co mě naučil, a také své rodině za podporu a toleranci.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	10
TEORETICKÁ ČÁST.....	11
1 RED HAT ENTERPRISE MRG MESSAGING	12
1.1 RED HAT ENTERPRISE MRG PROJEKT	12
1.2 ADVANCED MESSAGE QUEUING PROTOCOL (AMQP).....	13
1.3 RED HAT ENTERPRISE MRG MESSAGING MODEL	13
1.4 MESSAGE BROKER	15
1.5 KLIENTI	15
1.6 EXCHANGE	16
1.7 BINDING	16
1.8 FRONTY ZPRÁV	16
1.9 ZPRÁVY	17
1.10 RELACE	17
1.11 ZABEZPEČENÍ	18
1.12 VYČÍSLENÍ VÝKONNOSTI	18
2 ZPRACOVÁNÍ NAMĚŘENÝCH DAT	19
2.1 PŘESNOST MĚŘENÍ.....	19
2.2 ODHAD VÝSLEDNÉ HODNOTY A NEJISTOTY MĚŘENÍ.....	20
2.3 ODLEHLÉ HODNOTY	21
2.3.1 Test $\pm 4s$	22
PRAKTICKÁ ČÁST	23
3 ANALÝZA ZADÁNÍ.....	24
3.1 SPECIFIKACE POŽADAVKŮ	25
4 IMPLEMENTACE	27
4.1 ANALÝZA APLIKACE.....	27
4.1.1 Třída QpidBenchmarkStats	28
4.1.2 Třída StatsProcessing	30
4.1.3 Třída DataProcessing	31
4.1.4 Třída DataStore	33
4.1.5 Třída Serialization	34
4.1.6 Diagram tříd	35
4.2 PRINCIP ČINNOSTI	36
4.3 STRUKTURA APLIKACE	38
4.3.1 „qpid_benchmark_stats.py“	39
4.3.2 „qpid_benchmark_stats_tools.py“	40
4.3.3 „compare_data.py“	41

4.3.4	„qpid_benchmark_mod.py“	43
5	UŽIVATELSKÁ PŘÍRUČKA	44
5.1	OBSAH PŘILOŽENÉHO DISKU	44
5.2	SYSTÉMOVÉ POŽADAVKY	44
5.3	INSTALACE A KONFIGURACE SLUŽBY „QPIDD“	46
5.4	STAŽENÍ, PŘEKLAD A KOMPILACE APLIKACÍ „QPID-SEND“ A „QPID-RECEIVE“	47
5.5	ROZBALENÍ A SPUŠTĚNÍ APLIKACE „QPID_BENCHMARK_STATS“	48
5.5.1	Rozbalení.....	48
5.5.2	Spuštění aplikace.....	48
5.5.3	Výstup	50
5.6	SERIALIZOVANÁ DATA A MOŽNOSTI JEJICH VYUŽITÍ	50
	ZÁVĚR	53
	CONCLUSION	54
	SEZNAM POUŽITÉ LITERATURY A INTERNETOVÉ ZDROJE	55
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	56
	SEZNAM OBRÁZKŮ	58
	SEZNAM PŘÍLOH.....	59

ÚVOD

Komunikace prostřednictvím zpráv má své kořeny již ve starověku. S nástupem internetu se rozšířila a dnes tvoří významnou část elektronických služeb. Pomocí zpráv spolu komunikují dnes nejen lidé (uživatelé), ale i samotné aplikace. Objem odesílaných dat neustále vzrůstá a zvyšují se nároky na rychlost a bezpečnost elektronické komunikace. Přední světoví dodavatelé softwaru se snaží držet krok s tímto trendem, proto každá nová verze aplikace i komunikačního protokolu by měla pracovat spolehlivěji a rychleji, to znamená s větší propustností dat (throughput) a s menší či lépe predikovatelnou latencí (latency).

Společnost Red Hat vyvíjí produkt Red Hat Enterprise MRG. Jedna z jeho částí je Messaging, platforma pro rychlé a bezpečné posílání zpráv po síti (více v dalším textu). Firma Red Hat klade velký důraz na výkonnost a spolehlivost svých produktů, proto každá nová verze před svým vydáním prochází velmi důkladným procesem testování a ladění, jehož nedílnou součástí je v případě projektu Red Hat Enterprise MRG Messaging i měření propustnosti a odezvy brokeru při posílání zpráv. K tomuto účelu používá aplikaci „qpids-cpp-benchmark.py“, dostupnou na webových stránkách Apache-SVN [1]. Cílem této diplomové práce je vyvinout softwarový nástroj, který bude tato naměřená data zpracovávat a archivovat.

I. TEORETICKÁ ČÁST

1 RED HAT ENTERPRISE MRG MESSAGING

Red Hat Enterprise MRG Messaging je součástí projektu Red Hat Enterprise MRG (Messaging, Realtime, Grid). Jde o vysoce výkonný softwarový nástroj pro rychlé, spolehlivé a bezpečné posílání zpráv po síti. Je založený na AMQP (Advanced Message Queuing Protocol) protokolu a poskytuje spolehlivý nástroj pro posílání zpráv podporující mimo jiné zasílání velkých objemů dat, vysokou dostupnost (*high availability*) a ukládání zpráv na disk (*message durability*). Základem Red Hat MRG Messagingu je projekt Apache Qpid distribuovaný pod licencí Apache (více informací v [2]).

1.1 Red Hat Enterprise MRG projekt

Red Hat Enterprise MRG je výkonná výpočetní platforma s distribuovanou architekturou skládající se ze tří komponent:

- *Red Hat Enterprise MRG Messaging* – nástroj pro rychlé, spolehlivé a bezpečné zasílání zpráv po síti (detailněji je popsán v následujícím textu).
- *Red Hat Enterprise MRG Realtime* – sada nástrojů pro snížení a stabilizaci odezvy za účelem zvýšení výkonnosti kernelového jádra, optimálně pracujícího v reálném čase.
- *Red Hat Enterprise MRG Grid* – vysoce výkonná a spolehlivá síťová platforma pro distribuované plánování a vykonávání úloh, založená na technologii „cloud computing“.

Každá z výše uvedených komponent je navržena k užívání jako součást platformy Red Hat Enterprise MRG, ale může být používána i samostatně.

Pro zvýšení uživatelského komfortu poskytuje Red Hat Enterprise MRG také grafické webové rozhraní *MRG Management Console*, nazývaný též *Cumin*, umožňující přístup přes běžný webový prohlížeč (například Mozilla Firefox).

Výpočetní platforma Red Hat Enterprise MRG je součástí operačního systému Red Hat Enterprise Linux Server 6 a je možné ji nainstalovat a používat i na operačním systému Red Hat Enterprise Linux Server 5. Některé části pak lze používat i na jiných operačních systémech, například na Microsoft Windows (více informací v [3]).

1.2 Advanced Message Queuing Protocol (AMQP)

Advanced Message Queuing Protocol (dále jen AMQP) je otevřený standard navržený na podporu posílání zpráv mezi aplikacemi s distribuovanou architekturou nebo business aplikacemi. Umožňuje flexibilní způsoby směřování zpráv, „peer-to-peer“, „fanout“, „publish-subscribe“ nebo „request-response“. AMQP zprávy mohou obsahovat text i strukturovaná binární data.

AMQP standard podporuje rovněž potvrzované doručování zpráv, lokální i distribuované transakce, autentizaci klienta na základě „Simple Authentication and Security Layer“ (SASL) a opakované doručování zpráv a obnovu v případě přerušení spojení na síti.

V současné době aktuální verze MRG Messaging 2.3 podporuje AMQP protokol verze 0-10 [2].

1.3 Red Hat Enterprise MRG Messaging model

AMQP definuje přenos zpráv na dvou vrstvách ISO/OSI modelu:

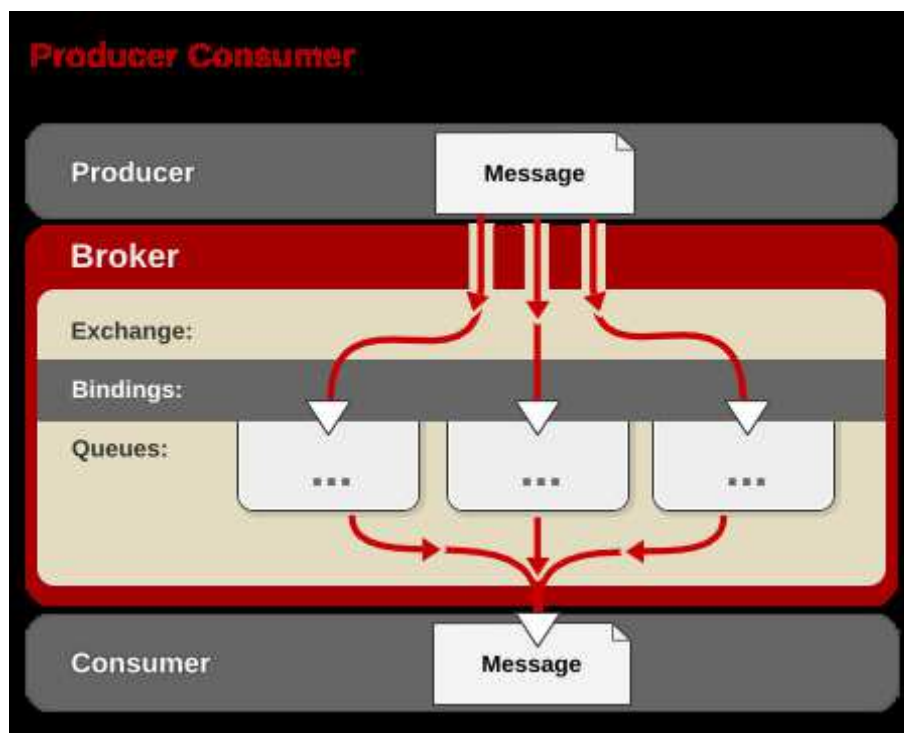
- na transportní vrstvě – protokol nižší úrovně;
- na aplikační vrstvě – protokol vyšší úrovně, definuje sémantická pravidla pro messaging.

MRG Messaging sestává zpravidla ze dvou typů aplikací:

- *Broker* – serverová aplikace, která poskytuje službu; může běžet jako aplikace nebo jako služba *qpidd*.
- *Klient* – aplikace, která využívá služeb brokeru, posílá (*producer*, *sender*) nebo přijímá (*consumer*, *receiver*) zprávy.

Připojením klienta k brokeru se otevře nová relace (*session*). Jejím prostřednictvím spolu klient a broker komunikují. Po skončení komunikace relace zpravidla zaniká. Relace může být synchronní i asynchronní.

Následující obrázek ukazuje, jak klienti *producer* a *consumer* s brokerem komunikují.



Obrázek 1: Komunikační model AMQP 0-10 [3]

Klient odesílatel (*producer*) vytvoří zprávu, naplní ji obsahem, volitelně k ní přidá směrovací klíč (*routing key*; řetězec, podle kterého může *exchange* určit, do kterých front má být zpráva doručena) a pošle zprávu do *exchange*. Před odesláním může odesílatel připojit ke zprávě vlastnosti (*properties*). Tyto vlastnosti blíže specifikují chování zprávy, například, je-li zpráva *durable*, MRG Messaging broker ji neztratí ani v případě hardwarového selhání.

Exchange přijme zprávu a pokud najde frontu (*queue*), která splňuje kritéria daná odesílatelem, předá ji této frontě. Tím vzniká vazba mezi *exchange* a frontou (*binding*), která pro danou *exchange* specifikuje které zprávy by měly být do dané fronty doručeny. Pokud pro *exchange* taková fronta neexistuje, nebude žádná zpráva z *exchange* doručena.

Fronta shromažďuje zprávy a doručuje je příjemcům (*consumer*), kteří jsou zaregistrováni k odběru zpráv z této fronty.

Klienti mohou vytvářet, sdílet, používat a odstraňovat fronty zpráv, pokud k tomu mají příslušná oprávnění, příjemce se navíc může registrovat k odběru zpráv. Fronty mohou mít definovány specifické vlastnosti, například *durable* fronty jsou vždy uloženy na disk před

restartem brokeru, exkluzivní (*exclusive*) fronty umožňují odběr zpráv jen jednomu příjemci.

Podporováno je také doručování zpráv prostřednictvím transakcí. Pokud je na straně odesílatele požadováno doručení skupiny zpráv (*group message*) prostřednictvím transakce, zprávy a potvrzení přijetí zpráv od příjemce jsou vyhodnoceny společně a všechny zprávy v transakci jsou doručeny úspěšně nebo nedoručeny.

1.4 Message broker

Komunikace prostřednictvím zpráv přímo mezi dvěma aplikacemi vyžaduje, aby obě aplikace byly současně připojené do sítě, navzájem znaly své komunikační rozhraní a navíc aby rychlost odesílání dat byla stejná jako rychlost přijímání dat.

Message broker (dále jen broker) řeší výše zmíněné problémy tím, že zprostředkovává nezávislou vrstvu mezi dvěma komunikujícími aplikacemi. Odesílatel nemusí mít žádné informace o příjemcích zpráv, broker přejímá zodpovědnost za správné doručení zpráv za něj. Navíc je vybaven vyrovnávací pamětí (*buffer*), která umožňuje komunikujícím aplikacím odesílat a přijímat data různou rychlostí.

Brokery mohou být spojeny do skupin pro zvýšení dostupnosti (*High Availability Cluster*). Klastř je skupina brokerů, které se chovají navenek jako jeden broker. Všechny brokery v klastřu mají stejné fronty, *exchange* i vazby mezi nimi (*bindings*). Změny na kterémkoli brokeru v klastřu způsobí téměř okamžitou změnu na všech ostatních brokerech v tomtéž klastřu.

Red Hat MRG Messaging také podporuje spojení brokerů do *federace* (*broker federation*). Ta umožňuje vytváření *message routes*, jejichž prostřednictvím mohou být některé zprávy z jednoho brokeru automaticky přesměrovány na jiný broker.

1.5 Klienti

Odesílající klient (*sender*) posílá zprávu do *exchange* na broker, ne přímo do fronty zpráv, proto může odesílat zprávy i bez informací o příjemci této zprávy. O zařazení do příslušné fronty se stará *exchange*. Odesílání zpráv na broker je tedy naprosto nezávislé na příjmu zpráv.

Klientské aplikace přijímají zprávy z brokeru prostřednictvím front zpráv.

1.6 Exchange

V modelu AMQP 0-10 klient odesílatel posílá zprávy do *exchange*, která je okamžitě předává příslušným frontám. Algoritmus výběru příslušné fronty se řídí typem *exchange* a vazbou mezi *exchange* a frontou zpráv (*binding*).

MRG Messaging definuje pět základních typů *exchange*:

- *Fanout* – nejjednodušší typ, přeposílá zprávy všem připojeným frontám.
- *Direct* – předává zprávy jen těm připojeným frontám, jejichž připojovací klíč (*binding key*) je identický s routovacím klíčem zpráv (*routing key*).
- *Default* – používá jméno fronty jako připojovací klíč. V modelu AMQP 0-10 je každá fronta automaticky připojena k *default exchange*.
- *Topic* – používá stejný algoritmus jako *direct exchange*, rozdíl je v tom, že používá jako routovací klíč více slov spojených „.“; například „usa.news“, „eu.weather“. Připojovací klíč pak může použít zástupné znaky „*“ (jedno libovolné slovo) a „#“ (jedno nebo více slov).
- *Headers* – umožňuje jednoduché dotazování na vlastnosti definované v hlavičkách zpráv. Na základě výsledků těchto dotazů vybere příslušné fronty.

Uživatel si může definovat vlastní typ *exchange*, *custom exchange* s vlastním vázacím algoritmem.

Red Hat Enterprise MRG Messaging podporuje *XML Exchanges*, které mohou směřovat zprávy podle uživatelských kritérií zapsaných v XML souboru.

1.7 Binding

Binding je předpis, který musí splňovat zprávy z *exchange*, aby mohly být přeposlány do dané fronty. Jeho obsah je ovlivněn typem *exchange* a příjemcem.

1.8 Fronty zpráv

Fronty zpráv jsou úložiště zpráv. Umožňují klientským aplikacím přijímajícím zprávy zaregistrovat se k odběru určitých zpráv.

Fronty přijímají zprávy předávané z *exchange* a uchovávají je do té doby, dokud je nevyzvedne příslušný příjemce. Klienti přijímající zprávy mohou prohledávat fronty a získávat z nich zprávy. Zprávy mohou být znovu vráceny do fronty pro opakované doručení nebo mohou být příjemcem smazány.

Jedna fronta může být sdílená více registrovanými příjemci (*subscribers*) nebo může mít jen jednoho zaregistrovaného příjemce, pak hovoříme o exkluzivní frontě (*exclusive queue*).

Odesílající klienti mohou vytvářet a vázat fronty na *exchange*, zpřístupňovat fronty přijímacím klientům nebo jen posílat zprávy do *exchange* a vytvoření fronty včetně jejího navázání na *exchange* ponechat na příjemci.

1.9 Zprávy

Zpráva vytvořená odesílacím klientem se skládá z obsahu zprávy a hlavičky. Obsah zprávy obsahuje vlastní informaci, kterou má zpráva přenést a hlavička popisuje zprávu. Obsahuje informace, kam by měla být zpráva směrována a jakým způsobem s ní má být nakládáno během přenosu.

Zprávy mohou být sdruženy do skupin (*message groups*). Odeslání nebo příjem této skupiny zpráv je vnímáno jako atomická operace. Zprávy jsou pak doručovány společně v rámci jedné transakce, to znamená, že zprávy a potvrzení přijetí zpráv od příjemce jsou vyhodnoceny společně a všechny zprávy v transakci jsou doručeny úspěšně nebo nedoručeny.

1.10 Relace

Relace (*session*) je jednoznačně identifikovaná konverzace mezi klientskou aplikací (odesílatelem nebo příjemcem) a *brokerem*. Nová relace vzniká po připojení klienta k *brokeru*. Jejím prostřednictvím spolu klient a broker komunikují. Ke komunikaci používají připojení (*connection*), přičemž několik různých relací může používat jedno připojení. Po ukončení komunikace relace zaniká.

1.11 Zabezpečení

AMQP protokol využívá k autentizaci klientů „Simple Authentication and Security Layer“ (SASL). Jde o platformu, která podporuje více autentizačních metod. Pro větší zabezpečení se doporučuje použít DIGEST-MD5 nebo GSSAPI (Kerberos). Metoda PLAIN je dostatečně bezpečná jen pokud je použita společně se SSL a metodu ANONYMOUS nelze mezi bezpečné metody autentizace zařadit vůbec.

Autorizace v Red Hat MRG Messagingu používá „Access Control List“ (ACL), který vymezuje, které akce jsou povolené pro jednotlivé uživatele.

1.12 Vyčíslení výkonnosti

Výkonnost *Message brokeru (performance)* je zpravidla charakterizována veličinami:

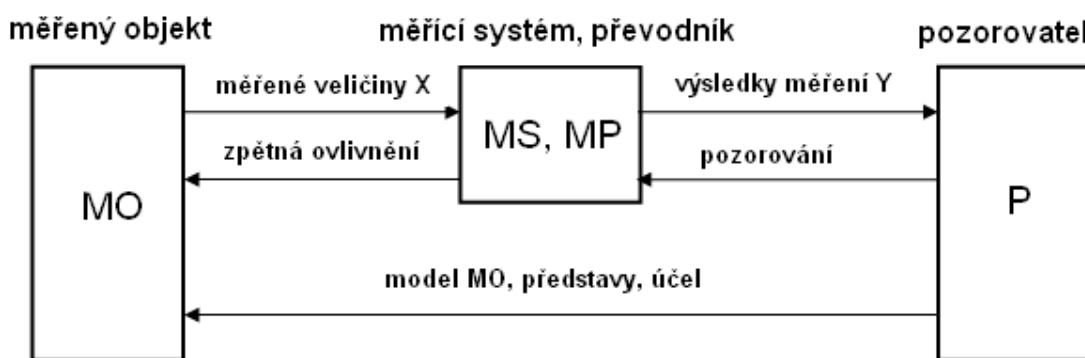
- propustnost (*throughput*) – množství přenesených dat za jednotku času; v této práci ji budeme udávat v jednotkách počet zpráv za sekundu nebo počet bytů za sekundu;
- latence zprávy (*latency*), někdy také odezva brokeru – doba od doručení zprávy na broker do jejího předání přijímacímu klientovi (zdržení zprávy na brokeru); udává se v jednotkách času, zde budeme používat milisekundy.

Výkonnost brokeru je závislá nejen na hardwarovém vybavení a operačním systému počítače, na kterém běží broker, ale také na propustnosti klientských aplikací.

2 ZPRACOVÁNÍ NAMĚŘENÝCH DAT

Proces měření lze obecně charakterizovat jako určitou transformaci souboru vstupních veličin X do souboru výstupních veličin Y , který představuje výsledek měření. V tomto modelu jsou obsaženy ovlivňující veličiny:

- připojení k měřenému objektu a tím jeho zpětné ovlivnění,
- vnější rušení měronosného signálu,
- nedostatky a chyby transformace vstupní veličiny na výstupní $X \rightarrow Y$,
- subjektivita pozorovatele (způsobená představami o měřeném objektu, tvorbě modelu objektu, apriorními informacemi atd.). [4]



Obrázek 2: Obecný model procesu měření [4]

2.1 Přesnost měření

Z výše uvedeného modelu je patrné, že každé měření je ovlivněno řadou faktorů, z nichž některé můžeme přesně vyčíslit, ale u velké části z nich to není možné, protože jde o náhodné veličiny. Navíc i samotné měření ovlivňuje měřenou veličinu. Z těchto důvodů přesnou hodnotu veličiny není možné změřit, proto k ní zpravidla přistupujeme jako k náhodné veličině a k určení její hodnoty využíváme statistické metody.

Pro zvýšení pravděpodobnosti určení dostatečně přesného výsledku pomocí statistických metod je třeba provést větší počet měření, za co nejméně odlišných podmínek (optimálně za stejných). Tyto hodnoty jsou více či méně rozptýlené, rozptýlení hodnot můžeme charakterizovat pomocí *nejistoty měření*.

2.2 Odhad výsledné hodnoty a nejistoty měření

Nejistota měření (výsledku měření) je takový nezáporný parametr, který charakterizuje rozptýlení hodnot přiřazených k měřené veličině na základě určité použité informace.

Standardní nejistoty měření se dělí na

- *Nejistoty typu A* (značené u_A), které jsou způsobeny většinou náhodnými chybami a určí se statistickou analýzou naměřených hodnot získaných za přesně definovaných podmínek měření.
- *Nejistoty typu B* (značené u_B) jsou způsobeny známými nebo odhadnutelnými příčinami. Příčin vzniku standardních nejistot hodnocených způsobem B může být více a výsledná standardní nejistota je dána jejich geometrickým součtem.

Nejistota měření typu A se získá statistickým vyhodnocením série opakovaných měření. Je-li n nezávislých pozorování provedeno za stejných podmínek, je odhad výsledné hodnoty prezentován hodnotou výběrového aritmetického průměru

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad (1)$$

Standardní nejistota typu A je pak určena výběrovou směrodatnou odchylkou (někdy též nazývanou „střední kvadratická chyba“) výběrového průměru.

$$u_{AX} = s(\bar{X}) = \sqrt{\frac{s^2(X_i)}{n}} = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^n (X_i - \bar{X})^2} \quad (2)$$

[4]

Tato skutečnost vyplývá z platnosti zákona velkých čísel a centrálních limitních vět [5], [6].

Standardní nejistoty typu B bývají zapříčiněny nedokonalostmi měřících prostředků, použitých metod měření, nestálostí podmínek apod. Při jejich určování se zpravidla vychází z racionálních úsudků. Výsledná nejistota typu B je pak dána jejich geometrickým součtem

$$u_{BX} = \sqrt{\sum_{i=1}^m u_{X,ZI}^2}, \quad (3)$$

kde $u_{X,ZI}$ jsou dílčí nejistoty typu B [4].

V této práci je k měření využíván softwarový nástroj. V souvislosti s určením standardní nejistoty měření typu B bychom měli diskutovat především o měření času prostřednictvím pythonového časovače „time.time()“. Pro naše účely je přesnost časovače dostatečná. Navíc výsledky měření se obvykle budou používat k vzájemným porovnáváním, proto můžeme tuto dílčí nejistotu zanedbat.

Dalším faktorem, který by mohl navýšit nejistotu typu B a ovlivnit výsledky měření, je fakt, že aplikace má určitou režii na svůj vlastní chod, zvýšenou zvláště na začátku, po svém spuštění. To znamená, že několik prvních měření by mohlo být ovlivněno větší měrou než zbývající měření. Pro minimalizaci této dílčí nejistoty měření aplikace „qpid_benchmark_stats.py“ umožňuje zpracovávat měření až od určitého pořadového čísla prostřednictvím vstupního parametru „--processed-from“. Nyní i tuto dílčí nejistotu měření typu B můžeme pro naše účely zanedbat.

Budeme tedy nadále uvažovat pouze standardní nejistotu měření typu A.

2.3 Odlehlé hodnoty

Odlehlými hodnotami nazýváme takové hodnoty, které se výraznou měrou liší svou velikostí od zbývajících skupiny výsledků v náhodném výběru. Tyto hodnoty mohou viditelně ovlivnit určení hodnoty výsledku. Proto je zpravidla ze skupiny naměřených hodnot vylučujeme ještě před určením odhadu výsledné hodnoty. Zde je však třeba dbát jisté opatrnosti, zvláště u menších počtů měření.

Při určování odlehlých hodnot pocházejících z jednoho výběru, je nejdříve uspořádáme podle velikosti

$$X_1 < X_2 < \dots < X_n$$

a testujeme hypotézu H_0 : naměřené hodnoty představují homogenní výběr ze souboru, který má normální rozdělení $N(\mu, \sigma^2)$, oproti hypotéze alternativní H_1 , která je zaměřená buď na největší hodnotu pozorování X_n , nejmenší hodnotu pozorování X_1 nebo na obě krajní hodnoty X_1 a X_n . [4]

Pro testování máme k dispozici různé metody, „Test $\pm 4s$ “, „Test Dixonův“, „Test Grubbsův“, „Test Shapiro-Wilksův“ a jiné. Zde popíšeme pouze „Test $\pm 4s$ “, který je

velmi jednoduchý, a proto byl vybrán pro implementaci v této aplikaci. Ostatní testy jsou podrobně popsány v [4].

2.3.1 Test $\pm 4s$

Tento test je vhodný pro použití u větších výběrů, které obsahují více než 10 hodnot. Po seřazení dat podle velikosti se za odlehlou hodnotu X_i^* pokládá ta krajní hodnota, která leží vně intervalu $\bar{X} \pm 4s$.

Aritmetický průměr \bar{X} počítáme podle vztahu (1) a odhad výběrové směrodatné odchylky podle vztahu

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2} \quad (4)$$

Hodnoty \bar{X} a s počítáme bez předpokládané odlehlé krajní hodnoty X_i^* [4].

II. PRAKTICKÁ ČÁST

3 ANALÝZA ZADÁNÍ

Testovací tým projektu Red Hat Enterprise MRG Messagingu v rámci testování svého produktu měří propustnost (*throughput*) a latenci zpráv (*latency*). Pro toto měření používá softwarový nástroj „qpidd-cpp-benchmark“, implementovaný v jazyce Python (ke stažení na stránkách Apache-SVN [1]). Ke své činnosti používá další dvě aplikace, „qpidd-send“ a „qpidd-receive“. Aplikace „qpidd-send“ je klientská aplikace odesílající zprávy a „qpidd-receive“ je klientská aplikace přijímající zprávy (Zdrojové kódy těchto aplikací v C++ jsou také k dispozici na [1]; podrobné informace o jejich stažení a kompilaci jsou v Uživatelské příručce, v kapitole 5). Samozřejmě je také zapotřebí mít připojení k běžícímu brokeru (aplikaci nebo službě qpidd, více informací v kapitole 1).

Aplikace „qpidd-cpp-benchmark“ pracuje následujícím způsobem (zjednodušený popis):

1. Spustí na pozadí nástroj „qpidd-receive“.
2. Spustí „qpidd-send“, pošle požadovaný počet zpráv.
3. Odečte výsledky: propustnost obou klientů (počet zpráv za sekundu), minimální, maximální a průměrnou hodnotu latence jedné zprávy v milisekundách.
4. Dopočítá celkovou propustnost (počet zpráv za sekundu).
5. Předchozí čtyři kroky opakuje podle zadaného počtu opakování (parametr „--repeat“).

Aplikace „qpidd-cpp-benchmark“ je spolehlivým a robustním nástrojem, navrženým a implementovaným špičkovými vývojáři komunity Apache, velice vhodným k získání výběrového souboru naměřených dat.

Dosud ovšem chyběl nástroj, který by dokázal tato získaná data nějakým vhodným způsobem zpracovat, to znamená určit na jejich základě pravděpodobné hodnoty naměřených veličin a chyby (případně nejistoty) měření. Toto je hlavním úkolem této diplomové práce.

3.1 Specifikace požadavků

Nová aplikace by měla splňovat tyto požadavky:

Funkční požadavky:

- Možnost zadávání více hodnot parametrů; aplikace by měla iterovat přes všechny jejich kombinace.
- Kontrola vstupních parametrů.
- Sledované veličiny:
 - propustnost odesílacího klienta (*sender throughput*) v počtu zpráv za sekundu,
 - propustnost přijímacího klienta (*receiver throughput*) v počtu zpráv za sekundu,
 - minimální latence zprávy (*minimum latency*) v milisekundách,
 - maximální latence zprávy (*maximum latency*) v milisekundách,
 - průměrná latence zprávy (*average latency*) v milisekundách,
 - celková propustnost (*total throughput*) v počtu zpráv za sekundu,
 - celková propustnost v počtu přenesených bytů za sekundu.
- Požadovaným výstupem pro výše uvedené veličiny je odhad výsledné hodnoty včetně odhadu nejistoty měření.
- Přehledný výstup ve formě tabulky.
- Uložení naměřených i vypočítaných dat v serializované formě do souboru pro možnost dalšího zpracování (například pro porovnání sad dat pocházejících z různých verzí produktu).

Nefunkční požadavky:

- Aplikace by měla být implementovaná v jazyce Python.

- Uživatelským rozhraním aplikace by měl být příkazový řádek (na většině testovacích strojů není nainstalované grafické uživatelské rozhraní).
- Požadováno je vypracování programové a uživatelské dokumentace.

4 IMPLEMENTACE

Vyvíjená aplikace bude ve většině případů používána na linuxovém operačním systému, ale v některých případech bude užitečné, aby mohla být použita i na jiném operačním systému, například na Microsoft Windows. Proto byl zvolen programovací jazyk Python [7].

Před vlastním vývojem aplikace bylo zapotřebí provést mírnou úpravu zdrojového kódu nástroje „qpid-cpp-benchmark“, používaného pro získání naměřených hodnot.

Aplikace „qpid-cpp-benchmark“ počítá celkovou propustnost v počtu zpráv za sekundu (*total throughput*) na základě předpokladu, že všechny odeslané zprávy v pořádku dojdou, nijak ovšem tuto skutečnost neověřuje. Navíc nevrací hodnotu propustnosti v bytech za sekundu. Pomocnou ruku při řešení tohoto problému mi podal můj konzultant a kolega, pan ing. František Řezníček, kterému touto cestou upřímně děkuji.

Upravená aplikace „qpid_benchmark_mod.py“ využívá k určení počtu odeslaných zpráv i objemu dat novou metodu „`get_queues_status()`“, která přímo monitoruje průchod dat frontami zpráv a vrací informace nejen o odeslaném a přijatém počtu zpráv, ale také o odeslaném a přijatém objemu dat v bytech a o nedoručených či chybně doručených zprávách. Do zdrojového kódu aplikace pak byla přidána kontrola, zda odeslaná data byla správně doručena a výpis informace o celkové propustnosti v bytech za sekundu.

Další drobnou změnou prošly vstupní parametry „`--qpid-send-path`“ a „`--qpid-receive-path`“. V původní aplikaci se jejich prostřednictvím zadává cesta k adresáři aplikací „qpid-send“ a „qpid-receive“ a při jejich spouštění se automaticky doplní tato jména aplikací.

V aplikaci „qpid_benchmark_mod.py“ jsou parametry „`--qpid-send-path`“ a „`--qpid-receive-path`“ cestami k aplikacím včetně jmen aplikací, počítá se s možností, že umístění nebo jména odesílacího a přijímacího klienta mohou být odlišná, například na operačním systému Microsoft Windows mají obvykle spustitelné soubory příponu „.exe“.

4.1 Analýza aplikace

Aplikace „qpid_benchmark_stats“, je implementovaná v objektově orientovaném paradigmatu; obsahuje pět tříd:

- QpidBenchmarkStats
- StatsProcessing
- DataProcessing
- DataStore
- Serialization

4.1.1 Třída QpidBenchmarkStats

Instance třídy obstarává hlavní funkcionalitu aplikace.

Metody:

run()

Načte vstupní parametry (*options*), ověří jejich formát, spustí „qpid-benchmark_mod.py“, přečte naměřené výsledky ze standardního výstupu, zpracuje je a vytiskne; nakonec naměřené i vypočítané hodnoty serializuje a uloží do souboru.

create_list()

Ze zadaného řetězce vytvoří a vrátí seříděný seznam bez duplicit .

Vstupní parametr:

- `opt_val` – řetězec hodnot oddělených čárkou.

convert_items_to_int()

Metoda se pokusí převést všechny prvky seznamu na typ „int“ a ověří, zda jejich hodnota spadá do požadovaného intervalu. Vrací seznam celočíselných hodnot v požadovaném intervalu.

Vstupní parametry:

- `in_list` – seznam hodnot,
- `min_val` – požadovaná minimální hodnota,
- `max_val` – požadovaná maximální hodnota.

convert_items_to_bool()

Pokusí se převést všechny prvky seznamu na typ „boolean“, vrací seznam hodnot typu „boolean“.

Vstupní parametr:

- `in_list` – seznam hodnot.

check_int_val()

Metoda ověřuje, zda daná hodnota `in_val` spadá do intervalu hodnot `<min_val, max_val>`, vrací výsledek (*True* nebo *False*).

Vstupní parametry:

- `in_val` – testovaná hodnota,
- `min_val` – požadovaná minimální hodnota (dolní hranice intervalu),
- `max_val` – požadovaná maximální hodnota (horní hranice intervalu).

get_datetime_string()

Vrací data ze struktury `datetime` v čitelné podobě.

Vstupní parametr:

- `dt` – proměnná typu „datetime“.

fill_in_data_list()

Vytvoří seznam, naplní ho daty s indexem „id“ pro statistické zpracování a vrátí ho. Seznam obsahuje dvojice (`<číslo měření>`, `<hodnota>`).

Vstupní parametry:

- `data_set` – seznam naměřených hodnot ze standardního výstupu aplikace „qpid_benchmark_mod.py“,
- `id` – číslo měření,
- `max_length` – maximální počet hodnot na řádku (= počet měřených veličin).

print_results()

Metoda vytiskne výsledky v přehledné tabulce.

Vstupní parametry:

- `tup_list` – seznam pětic vypočítaných hodnot k tisku (<průměrná hodnota>, <směrodatná odchylka průměru>, <počet platných hodnot>, <minimální hodnota>, <maximální hodnota>)

`print_footer()`

Metoda vytiskne všeobecné informace vztahující se k měření.

Vstupní parametry:

- `b_info` – slovník obsahující informace k tisku

4.1.2 Třída `StatsProcessing`

Objekt třídy provádí základní statistické výpočty (aritmetický průměr, rozptyl, výběrová směrodatná odchylka, výběrová směrodatná odchylka průměru, nalezení odlehlých hodnot).

Atribut:

`values` – seznam

Metody:

`__init__()`

Konstruktor přijímá jeden parametr, seznam hodnot ke zpracování. Hodnoty překonvertuje na typ „float“ a vytvoří z nich uspořádaný seznam, který uloží do proměnné `self.values`.

Vstupní parametr:

- `values` – seznam hodnot ke zpracování

`remove_negs()`

Metoda odstraní záporné hodnoty ze seznamu `self.values`.

`get_values()`

Metoda vrací seznam zpracovávaných hodnot, `self.values`.

`compute_mean()`

Metoda vrací aritmetický průměr vypočtený z hodnot seznamu `self.values`.

compute_scattering()

Metoda vrací výběrový rozptyl hodnot `self.values`.

compute_standard_deviation()

Metoda vrací výběrovou směrodatnou odchylku vypočtenou z hodnot seznamu `self.values`.

compute_mean_std_dev()

Metoda vrací výběrovou směrodatnou odchylku průměrné hodnoty seznamu `self.values`.

is_extreme()

Testuje metodou „Test $\pm 4s$ “ (popis v kapitole 2.3.1), zda je daná hodnota odlehlá v souboru hodnot ze seznamu `self.values`. Vrací výsledek testu (*True* nebo *False*).

Vstupní parametr:

- `value` – testovaná hodnota

find_extreme_values()

Vrací seznam odlehlých hodnot; testuje na odlehlost střídavě nejmenší a největší hodnotu seznamu `self.values`. Seznam odlehlých hodnot má délku nejvýše rovnou `max_extremes`.

Vstupní parametr:

- `max_extremes` – číslo udávající maximální povolený počet identifikovaných odlehlých hodnot.

get_minimum()

Vrací minimální hodnotu ze seznamu `self.values`.

get_maximum()

Vrací maximální hodnotu ze seznamu `self.values`.

4.1.3 Třída **DataProcessing**

Objekt třídy `DataProcessing` vytváří objekt třídy `StatsProcessing`, upravuje a zadává naměřené hodnoty ke zpracování tomuto objektu, vyzvedává výsledky.

Atribut:

`dlists` – seznam

Metody:

`__init__()`

Konstruktor uloží seznam `data_lists` do proměnné `self.dlists`.

Vstupní parametr:

- `data_lists` – seznam dvojic (<číslo měření>, <hodnota>)

`remove_measurements()`

Odstraní ze seznamu `self.dlists` měření s danými čísly.

Vstupní parametr:

- `numbers` – seznam čísel měření, která mají být odstraněna

`get_meas_numbers()`

Metoda vrací čísla měření, která obsahují dané hodnoty veličin.

Vstupní parametry:

- `it_id` – číslo veličiny,
- `values` – seznam hledaných hodnot specifikované veličiny.

`get_values_to_process()`

Vrací seznam hodnot pro zpracování (druhé složky dvojic).

Vstupní parametr:

- `i_list` – seznam dvojic (<číslo měření>, <hodnota>) s naměřenými hodnotami jedné veličiny.

`remove_extreme_measurements()`

Metoda odstraní měření obsahující odlehle hodnoty ze seznamu `self.dlists`, vrací seznam `self.dlists`.

Vstupní parametr:

- `rm_ext` – udává maximální počet hodnot pro veličinu, které mohou být identifikované jako odlehlé.

`get_stats()`

Metoda vytvoří objekt třídy `StatsProcessing`, předá mu data ke zpracování a vrátí seznam pětic výsledků zpracování (<průměr>, <směrodatná odchylka průměru>, <počet zpracovávaných hodnot>, <minimum>, <maximum>).

4.1.4 Třída `DataStore`

Objekt třídy vytvoří datové struktury pro uložení dat při serializaci, načte data z jedné sady měření a uloží je do vytvořených struktur.

Atribut:

`data` – slovník.

Metody:

`__init__()`

Konstruktor vytvoří prázdné datové struktury (slovníky).

`add_cmd_options()`

Najde a přidá parametry provedeného příkazu „`qpid_benchmark_mod.py`“.

Vstupní parametr:

- `cmd_opts` – seznam parametrů příkazu.

`add_measured_values()`

Metoda najde a přidá naměřené hodnoty.

Vstupní parametr:

- `m_values` – seznam hodnot.

`add_summary()`

Najde a přidá vypočítané hodnoty.

Vstupní parametr:

- `sum` – sada výsledků, seznam pětic s vypočítanými hodnotami (<průměr>, <směrodatná odchylka průměru>, <počet zpracovávaných hodnot>, <minimální hodnota>, <maximální hodnota>).

`get_data()`

Metoda vrací hodnotu proměnné `data`.

4.1.5 Třída `Serialization`

Objekt třídy vytvoří základní datovou strukturu (seznam) pro archivaci dat, sbírá data, doplní základní informace o provedeném měření a tato data serializuje, případně otevře existující soubor s daty, data deserializuje a vrací seznam s uloženými daty.

Atributy:

`data_container` – seznam,

`start_time` – proměnná typu „datetime“.

Metody:

`__init__()`

Konstruktor vytvoří datovou strukturu pro ukládání dat a uloží aktuální čas.

`get_cpu_count()`

Metoda vrací počet CPU na aktuálním stroji.

`add_basic_info()`

Metoda přidá základní informace o aktuálním stroji, čase měření a doplní popis měření (`description`).

Vstupní parametr:

- `description` – řetězec popisující provedené měření.

`add_data()`

Metoda přidá sadu dat.

Vstupní parametr:

- `data_store` – sada dat z jedné sady měření uložená objektem třídy `DataStore`.

`get_data()`

Přístupová metoda k proměnné `data_container`; vrací její hodnotu.

`serialize()`

Metoda serializuje data do souboru.

Vstupní parametry:

- `store_file` – základ názvu souboru serializovaných dat,
- `store_dir` – cesta k adresáři se soubory serializovaných dat.

`deserialize()`

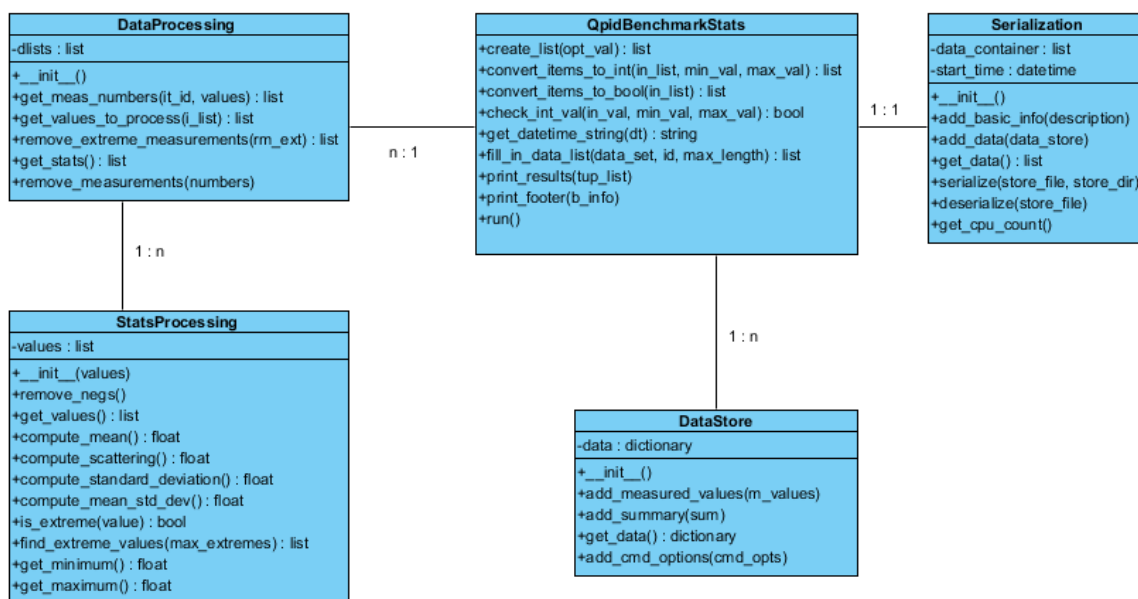
Metoda deserializuje data ze souboru.

Vstupní parametr:

- `store_file` – cesta k souboru s daty.

4.1.6 Diagram tříd

Třída `QpidBenchmarkStats` je ve vztahu asociace se třídami `DataProcessing`, `DataStore` a `Serialization`, třída `DataProcessing` je v asociaci ještě se třídou `StatsProcessing`, jak znázorňuje diagram tříd na následujícím obrázku.



Obrázek 3: Diagram tříd

4.2 Princip činnosti

Jak již bylo zmíněno, aplikace „qpid_benchmark_stats.py“ zpracovává data naměřená aplikací „qpid_benchmark_mod.py“.

Činnost nové aplikace „qpid_benchmark_stats.py“ probíhá v těchto krocích:

1. krok (příprava):

Metoda `main()` vytvoří objekt třídy `QpidBenchmarkStats` a zavolá jeho metodu `run()`. Metoda `run()` načte vstupní parametry (*options*), zkontroluje je a sestaví z nich seznamy pro iterace.

Vytvoří objekt třídy `Serialization` pro uložení a serializaci dat.

2. krok (měření):

Metoda `run()` vytvoří instanci třídy `DataStore`.

Na základě načtených parametrů sestaví a zobrazí na standardní výstup tvar prováděného příkazu „qpid_benchmark_mod.py“, atributy příkazu uloží do instance třídy `DataStore`.

Spustí „qpid_benchmark_mod.py“ s upravenými parametry načtenými ze vstupu; počet opakování (vstupní parametr „--repeat“) zvětší o počet počátečních měření, která

budou odstraněna, abychom minimalizovali vliv samotné aplikace na hodnoty naměřených dat (podrobné vysvětlení v kapitole 2.2).

Parametr „--repeat“ aplikace „qpid_benchmark_mod.py“ bude mít tedy hodnotu:

$N = \text{hodnota parametru „--repeat“} + (\text{hodnota parametru „--processed-from“} - 1)$.

To znamená, že „qpid_benchmark_mod.py“ N -krát odešle a přijme požadovaný (vždy stejný) počet zpráv za (téměř) stejných podmínek a na svém standardním výstupu vrátí data, která přečte naše aplikace a bude je dále zpracovávat. Získali jsme tímto v našem případě sedm souborů naměřených dat, pro každou ze sledovaných veličin jeden soubor.

Sledované veličiny jsou:

- propustnost odesílacího klienta (*sender throughput*) v počtu zpráv za sekundu,
- propustnost přijímacího klienta (*receiver throughput*) v počtu zpráv za sekundu,
- minimální latence zprávy (*minimum latency*) v milisekundách,
- maximální latence zprávy (*maximum latency*) v milisekundách,
- průměrná latence zprávy (*average latency*) v milisekundách,
- celková propustnost v počtu zpráv za sekundu (*total throughput in messages per second*),
- celková propustnost v bytech za sekundu (*total throughput in bytes per second*).

V dalším textu budeme pojmem „jedno měření“ označovat množinu dat získanou z jednoho měření, tedy minimálně sedmici naměřených hodnot (v případě, že vytváříme více front zpráv či všeobecně více odesílacích nebo přijímacích klientů, počet hodnot v jednom měření se zvyšuje).

3. krok (odstranění prvních měření):

Naměřené hodnoty získané ze standardního výstupu „qpid_benchmark_mod.py“ aplikace ukládá do své vnitřní proměnné, po úpravách je uloží do objektu třídy `DataStore`.

Vytvoří instanci třídy `DataProcessing`, zavolá její metodu

`remove_measurements()`, která odstraní požadovaný počet prvních měření.

4. krok (nalezení a vyloučení odlehlých hodnot):

Metoda `remove_extreme_measurements()` objektu třídy `DataProcessing` vytvoří objekt třídy `StatsProcessing`, jehož metoda `find_extreme_values()` dokáže najít odlehlé hodnoty.

Pro každou z měřených veličin zvlášť testuje krajní hodnoty na odlehlost metodou „Test $\pm 4s$ “ (podrobný popis algoritmu v kapitole 2.3.1). Počet hodnot, které mohou být identifikovány jako odlehlé, je limitován parametrem „--max-removed-extremes“. Odlehlé hodnoty vrátí, objekt třídy `DataProcessing` najde příslušná měření, za kterých tyto hodnoty pocházejí, a odstraní celá měření.

5. krok (výpočet aritmetického průměru a výběrové směrodatné odchylky průměru):

Objekt třídy `DataProcessing` vytvoří nový objekt třídy `StatsProcessing` a spočítá pro každou veličinu aritmetický průměr jako odhad výsledné naměřené hodnoty podle vztahu (1) a výběrovou směrodatnou odchylku průměru jako standardní nejistotu měření podle vztahu (2) (podrobné vysvětlení je v kapitole 2.2), dále počet hodnot, ze kterých byly statistické veličiny počítány (tj. počet naměřených hodnot zmenšený o odstraněná první měření a odlehlé hodnoty) a minimální a maximální hodnotu.

Výsledky vypíše na standardní výstup a uloží do objektu třídy `DataStore`.

Data spravovaná objektem třídy `DataStore` předá objektu třídy `Serialization`.

6. krok (iterace přes zadané parametry):

Kroky 2 – 5 opakuje pro všechny kombinace zadaných vstupních parametrů.

7. krok (serializace):

Instance třídy `QpidBenchmarkStats` předá objektu třídy `Serialization` informace o stroji, čase a měření. Objekt třídy `Serialization` data serializuje a uloží do souboru.

4.3 Struktura aplikace

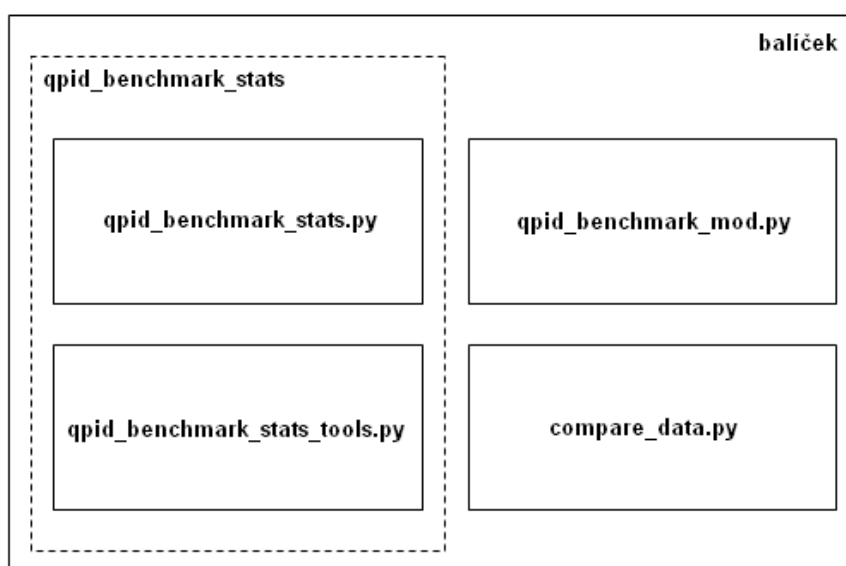
Aplikace „qpid_benchmark_stats“ sestává ze dvou modulů:

- „qpid_benchmark_stats.py“ – spustitelný modul obsahující metodu `main()` a třídu `QpidBenchmarkStats`.

- „qpid_benchmark_stats_tools.py“ – modul, který je třeba importovat; obsahuje definice tříd s metodami, které spustitelný modul využívá.

Do balíčku jsou přidány ještě dva spustitelné moduly:

- „qpid_benchmark_mod.py“ – upravená aplikace “qpid-cpp-benchmark“ používaná pro získání souboru dat,
- „compare-data.py“ – ukázková aplikace využití uložených výsledků; umí vypsat serializovaná data ze souborů a porovnat je podle zadaných kritérií.



Obrázek 4: Obsah balíčku „qpid_benchmark_stats.zip“

V další části kapitoly jsou stručně popsány jednotlivé moduly. Vygenerovanou dokumentaci ze zdrojových kódů (v anglickém jazyce) najdete v příloze P I.

4.3.1 „qpid_benchmark_stats.py“

Spustitelný modul aplikace.

Spouští „qpid_benchmark_mod.py“, zpracovává měření ze standardního výstupu a vypisuje informace o hodnotách veličin:

Obsah modulu:

- třída `QpidBenchmarkStats` - třída je popsána v kapitole 4.1.1

- metoda `main()`

Metoda `main()`

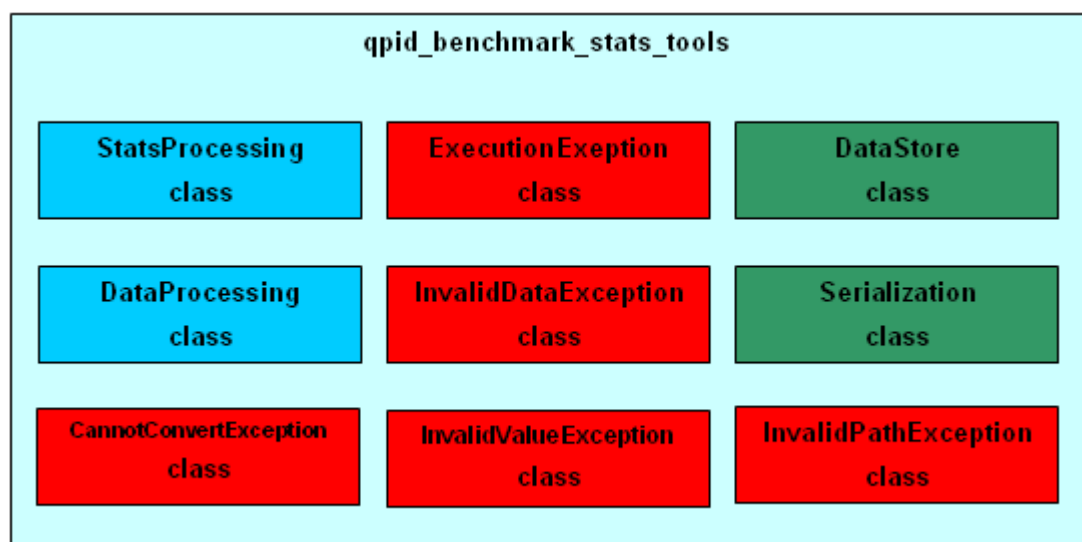
Metoda `main()` vytvoří instanci třídy `QpidBenchmarkStats` a zavolá její metodu `run()`.

4.3.2 „qpid_benchmark_stats_tools.py“

Modul obsahuje třídy poskytující metody pro formátování, statistické zpracování, serializaci dat a definice tříd vyjímek.

Obsah modulu:

- třída `StatsProcessing` – třída je popsána v kapitole 4.1.2.
- třída `DataProcessing` - třída je popsána v kapitole 4.1.3.
- třída `DataStore` - třída je popsána v kapitole 4.1.4.
- třída `Serialization` - třída je popsána v kapitole 4.1.5.
- třídy vyjímek



Obrázek 5: Obsah modulu „qpid_benchmark_stats_tools“

Třídy vyjímek – pro odlišení příčiny vzniku vyjímky

- **ExecutionException** – nastává, pokud volaná aplikace skončí vyjímku,

- **InvalidDataException** – nastává, pokud požadovaná operace nemůže být na daných datech provedena,
- **InvalidValueException** – nastává, pokud má proměnná nevyhovující hodnotu,
- **CannotConvertException** – nastává, pokud selže operace konverze dat,
- **InvalidPathException** – nastává, pokud specifikovaná cesta (soubor, adresář) neexistuje.

4.3.3 „compare_data.py“

Aplikace demonstrující deserializaci, zobrazení a porovnání uložených dat ze dvou souborů „*.data“.

Vstupní parametry (*options*):

- `-f, --file` – cesta k souboru (souborům),
- `-o, --identical-options` – vstupní parametry příkazu „qpid-benchmark_stats_mod“, které mají být pro dvojici příkazů identické při vzájemném porovnávání,
- `--print-data` – pokud má parametr hodnotu *True*, zobrazí deserializovaný obsah souboru.

Metody:

main()

Přečte vstupní parametry, ověří, deserializuje data a vytiskne je. Najde porovnatelná data (podle zadaných kritérií) a porovnatelné dvojice vytiskne.

print_basic_info()

Metoda vytiskne základní informace o uloženém měření (o stroji, čase apod.).

Vstupní parametr:

- `data` – seznam (deserializovaná data).

print_command()

Metoda vytiskne tvar provedeného příkazu „qpid_benchmark_mod.py“.

Vstupní parametr:

- `i_dict` – slovník s daty.

`print_summary()`

Metoda vytiskne souhrnné vypočítané hodnoty pocházející z provedení příkazu „`qpid_benchmark_mod.py`“ s jednou n-ticí parametrů.

Vstupní parametr:

- `i_dict` – slovník s daty.

`print_content_of_data_file()`

Metoda vytiskne deserializovaná data (obsah datového souboru v čitelné formě).

Vstupní parametr:

- `data` – seznam (deserializovaná data).

`is_comparable()`

Metoda porovná parametry příkazu „`qpid_benchmark_mod.py`“ ve slovnících `dict1` a `dict2` a vrátí výsledek.

Vstupní parametry:

- `dict1` – slovník s daty z prvního souboru naměřenými s jednou n-ticí parametrů příkazu „`qpid_benchmark_mod.py`“,
- `dict2` – slovník s daty z druhého souboru naměřenými s jednou n-ticí parametrů příkazu „`qpid_benchmark_mod.py`“,
- `i_opts` – seznam klíčů (jmen parametrů), které by měly být identické, aby měření byla srovnatelná.

`get_comparable_data()`

Vrátí seznam seznamů srovnatelných dat.

Vstupní parametry:

- `data1` – seznam, deserializovaná data z prvního souboru,
- `data2` – seznam, deserializovaná data z druhého souboru ,

- `i_opts` - seznam klíčů (jmen parametrů), které by měly být identické, aby měření byla srovnatelná.

`print_results()`

Metoda vytiskne srovnatelné výsledky.

Vstupní parametr:

- `c_data` – seznam seznamů s porovnatelnými daty.

4.3.4 „`qpid_benchmark_mod.py`“

Mírně upravený původní modul „`qpid-cpp-benchmark`“. Více informací na začátku kapitoly 4 a v dokumentaci (příloha P I).

5 UŽIVATELSKÁ PŘÍRUČKA

5.1 Obsah přiloženého disku

Příbalený disk obsahuje tři adresáře:

- DOC – obsahuje soubory:
 - „README“ – základní instrukce pro instalaci a použití,
 - „LICENCE“ – licenční ujednání,
 - „fulltext.pdf“ – text diplomové práce.
- SRC – obsahuje balíček „qpid_benchmark_stats.zip“ se zdrojovými kódy aplikace:
 - „qpid_benchmark_stats.py“,
 - „qpid_benchmark_stats_tools.py“,
 - „qpid_benchmark_mod.py“,
 - „compare_data.py“.
- DATA – obsahuje:
 - soubor „benchmark_stats_stdout.log“ - ukázka výstupu programu „qpid_benchmark_stats.py“,
 - adresář „data“ se serializovanými daty,
 - soubor „compare_stdout.log“ – ukázka výstupu programu „compare_data.py“.

5.2 Systémové požadavky

Aplikace „qpid_benchmark_stats.py“ běží na klientském počítači, ale vyžaduje přístup k běžící službě „qpidd“; je tedy třeba, aby na počítači, ke kterému se bude připojovat (může jít i o tentýž stroj), byl nainstalovaný a spuštěný Red Hat Enterprise Messaging broker.

Dále aplikace „qpid_benchmark_stats.py“ spouští prostřednictvím volání aplikace „qpid_benchmark_mod.py“ dvě klientské aplikace „qpid-send“, který posílá zprávy na broker a „qpid-receive“, který zprávy přijímá. Aplikace „qpid-send“ a „qpid-receive“

mohou být obecně implementované v různých programovacích jazycích a mohou běžet na operačním systému Red Hat Enterprise Linux 5 nebo 6 nebo na systémech Microsoft Windows XP, 7, Server 2003 nebo Server 2008.

Požadavky na server (stroj, kde poběží Red Hat Enterprise MRG Messaging Broker)

- Operační systém Red Hat Enterprise Linux 5 nebo 6.
- Red Hat Enterprise MRG Messaging verze 2.2 (qpidd-0.14-22) nebo vyšší.

Požadavky na klientský stroj (zde poběží aplikace „qpidd_benchmark_stats“):

- Operační systém:
 - Red Hat Enterprise Linux 5 nebo 6
 - Microsoft Windows XP, 7, Server 2003 nebo Server 2008
- Python verze 2.4 nebo vyšší.
- Nástroj na rozbalení balíčků - unzip.
- Překladač a kompilátor programovacího jazyka C++ (například GNU Compiler Collection (GCC)) pro kompilaci klientů „qpidd-send“ a „qpidd-receive“ implementovaných v jazyce C++ na linuxové platformě.

Výše zmíněný nástroj nainstalujeme pod účtem „root“ příkazem:

```
yum install gcc-c++
```

Poznámka:

Operační systém Red Hat Enterprise Linux 6 již obsahuje základní verzi Red Hat Enterprise Messaging.

Pro vyzkoušení aplikace je možné po založení uživatelského účtu stáhnout 30-denní trial verzi operačního systému Red Hat Enterprise Linux 6.4 z webových stránek společnosti Red Hat [8].

Aplikaci „qpidd-benchmark-stats“ lze rovněž spustit na operačním systému CentOS verze 6.4, který obsahuje „qpidd“ balíčky, kompatibilní s „qpidd“ balíčky, které jsou součástí Red Hat Enterprise MRG Messagingu. Naměřené výsledky se ovšem mohou značně lišit a nejsou vyloučeny ani drobné odlišnosti v chování aplikace. Službu „qpidd“ na stroj s operačním systémem CentOS nainstalujeme například příkazem:

```
yum install "*qpidd*"
```

Při konfiguraci postupujeme stejně jako v případě Red Hat MRG Enterprise Messagingu, postup je popsán v následující kapitole.

5.3 Instalace a konfigurace služby „qpidd“

Při instalaci služby „qpidd“ (Red Hat Enterprise MRG Messaging brokeru) postupujeme podle návodu v dokumentu „Messaging Installation and Configuration Guide“ ke stažení na stránkách [3].

Pokud se spokojíme s verzí Messagingu, který je součástí Red Hat Enterprise Linuxu 6, postupujeme tímto způsobem:

Po registraci do RHN systému nainstalujeme MRG Messaging a balíček „qpidd-tools“, potřebný pro správnou funkci nástroje „qpidd_benchmark_mod.py“ příkazy:

```
yum groupinstall "MRG Messaging"  
yum install qpidd-tools
```

Po dokončení instalace otevřeme konfigurační soubor „/etc/qpidd.conf“ a upravíme konfiguraci brokeru.

Pokud nám primárně nejde o testování autentizace, můžeme ji pro testovací účely vypnout. Velmi užitečné je nastavení logování do souboru a požadované úrovně logování (na výpise dole je nastavena úroveň info+).

Doporučená konfigurace:

```
cat /etc/qpidd.conf  
auth=no  
log-to-file=/var/lib/qpidd/qpidd.log  
log-enable=info+
```

Po úpravě konfiguračního souboru je třeba službu „qpidd“ restartovat:

```
service qpidd restart
```

Nakonec upravíme nastavení firewallu a službu „iptables“ restartujeme:

```
iptables -I INPUT -p tcp -m tcp --dport 5672 -j ACCEPT  
service iptables save  
service iptables restart
```

5.4 Stažení, překlad a kompilace aplikací „qpid-send“ a „qpid-receive“

Klientské aplikace „qpid-send“ a „qpid-receive“ mohou být implementovány v jakémkoli z programovacích jazyků, podporovaném platformou Red Hat Enterprise MRG Messaging (výčet aktuálně podporovaných platforem najdete v dokumentaci [3]).

Pro praktické měření propustnosti a odezvy brokeru se nejčastěji používají klienti implementovaní v programovacím jazyce C++ na linuxové platformě.

Od verze Red Hat Enterprise MRG 2.3 (qpid-0.18-*) jsou spustitelné aplikace „qpid-send“ a „qpid-receive“ implementované v jazyce C++ součástí rozšířeného instalačního balíčku. Obvykle jsou umístěné v adresáři „/opt/rh-qpid/clients/“. Pokud aplikace „qpid-send“ a „qpid-receive“ k dispozici nejsou, je třeba je stáhnout, přeložit a zkompilovat.

Z webové stránky Apache-SVN [1] stáhneme do pracovního adresáře (doporučené umístění: „/opt/rh-qpid/clients/“; pokud neexistuje, vytvoříme jej) soubory se zdrojovými kódy:

- qpid-send.cpp
- qpid-receive.cpp
- Statistics.cpp
- Statistics.h
- TestOptions.h
- ConnectionOptions.h

Přesuneme se do pracovního adresáře a následujícími příkazy pod uživatelským účtem „root“ přeložíme a zkompilujeme klienty „qpid-send“ a „qpid-receive“:

```
# přesun do pracovního adresáře
cd /opt/rh-qpid/clients
# překlad a kompilace
g++ -Wall -I. Statistics.cpp qpid-send.cpp -o qpid-send -lqpidmessaging
g++ -Wall -I. Statistics.cpp qpid-receive.cpp -o qpid-receive -lqpidmessaging
# přidáme oprávnění ke spuštění aplikací
chmod a+x qpid-send
chmod a+x qpid-receive
```

Ověříme, že v pracovním adresáři nám přibýly spustitelné soubory „qpid-send“ a „qpid-receive“.

5.5 Rozbalení a spuštění aplikace „qpid_benchmark_stats“

5.5.1 Rozbalení

Balíček „qpid_benchmark_stats.zip“ umístíme do pracovního adresáře (pracovní adresář nemusí být totožný s adresářem, kde jsou umístěné aplikace „qpid-send“ a „qpid-receive“). Přesuneme se do pracovního adresáře a balíček rozbalíme pomocí utility „unzip“:

```
unzip qpid_benchmark_stats.zip
```

V pracovním adresáři nám přibyla složka „qpid_benchmark_stats“.

Vstoupíme do ní a uvnitř najdeme soubory:

- „qpid_benchmark_stats.py“ – spustitelný soubor,
- „qpid_benchmark_stats_tools.py“ – importovaný modul,
- „qpid_benchmark_mod.py“ – aplikace, kterou „qpid_benchmark_stats.py“ spouští,
- „compare_data.py“ – spustitelný soubor demonstrující zobrazení a využití serializovaných dat.

5.5.2 Spuštění aplikace

Aplikaci „qpid_benchmark_stats.py“ spustíme příkazem:

```
python qpid_benchmark_stats.py [option1] [option2] ...
```

Pro zobrazení všech argumentů (*options*) můžeme zavolat příkaz s argumentem „--help“.

```
python qpid_benchmark_stats.py --help
```

Přehled argumentů a jejich význam:

- -b, --broker – URL adresa brokeru, je možné zadat více adres oddělených čárkami bez mezer; výchozí hodnota „127.0.0.1“,
- --qpid-send-path – cesta k aplikaci „qpid-send“, výchozí hodnota „/opt/rh-qpid/clients/qpid-send“,

- `--qpuid-receive-path` – cesta k aplikaci „qpuid-receive“, výchozí hodnota „`/opt/rh-qpuid/clients/qpuid-receive`“,
- `--connection-options` – „connection options“ (řetězec blíže specifikující připojení) klientských aplikací k brokeru,
- `--queue-name` – jméno fronty zpráv, výchozí hodnota „benchmark“,
- `-q`, `--queues` – počet front zpráv, lze zadat i více hodnot oddělených čárkami; výchozí hodnota „1,2“,
- `-s`, `--senders` – počet vytvořených odesílacích klientů, lze zadat i více hodnot oddělených čárkami; výchozí hodnota „1,2“,
- `-r`, `--receivers` – počet vytvořených přijímacích klientů, lze zadat i více hodnot oddělených čárkami; výchozí hodnota „1,2“,
- `-m`, `--messages` – počet zpráv, lze zadat i více hodnot oddělených čárkami; výchozí hodnota „100,1000,10000“,
- `--content-size` – velikost jedné zprávy v bytech, lze zadat i více hodnot oddělených čárkami; výchozí hodnota „1024“,
- `--ack-frequency` – počet zpráv, po kterých se potvrzuje doručení, lze zadat i více hodnot oddělených čárkami; výchozí hodnota „100“,
- `--durable` – trvanlivost zprávy (*durability*), lze zadat i více hodnot oddělených čárkami; výchozí hodnota „False,True“,
- `--max-removed-extremes` – maximální počet odlehlých hodnot, který může být odstraněn (pro jednu veličinu), výchozí hodnota 10% počtu měření,
- `--processed-from` – pořadové číslo prvního zpracovávaného měření (předchozí měření se zahazují), výchozí hodnota 6,
- `--fill-drain` – pokud je „True“, zprávy se odesílají až po naplnění front, výchozí hodnota „False“,
- `--repeat` – počet měření (pro jednu kombinaci parametrů), výchozí hodnota 50,
- `--store-file` – jméno souboru pro uložení dat, výchozí hodnota „store_file“, jméno souboru má tvar `<jméno_souboru>_<rok>_<měsíc>_<den>_<čas_uložení>`,

- `--description` – popis měření,
- `--store-directory` – jméno adresáře pro uložení dat, výchozí hodnota „data“.

5.5.3 Výstup

Aplikace „`qpid_benchmark_stats.py`“ vrací na standardním výstupu informace o naměřených hodnotách veličin:

- `send-tp` – propustnost klienta „`qpid-send`“ v počtu zpráv za sekundu,
- `recv-tp` – propustnost klienta „`qpid-receive`“ v počtu zpráv za sekundu,
- `l-max` – maximální naměřená latence jedné zprávy v milisekundách,
- `l-min` – minimální naměřená latence jedné zprávy v milisekundách,
- `l-avg` – průměrná naměřená latence jedné zprávy v milisekundách,
- `tp-msg` – celková propustnost v počtu zpráv za sekundu,
- `tp-byte` – celková propustnost v bytech za sekundu.

Pro každou z těchto veličin vrací:

- `mean` – aritmetický průměr naměřených hodnot,
- `mean std dev` – směrodatná odchylka průměru (vyčíslená standardní nejistota)
- `valid values` – počet hodnot, se kterými je počítáno (počet hodnot po odstranění odlehlých hodnot),
- `min value` – minimální naměřená hodnota (po odstranění odlehlých hodnot),
- `max value` – maximální naměřená hodnota (po odstranění odlehlých hodnot).

Příklad standardního výstupu aplikace je v příloze P II.

5.6 Serializovaná data a možnosti jejich využití

Soubory „`*.data`“ uchovávají serializovaná data aplikace „`qpid_benchmark_stats`“.

Deserializaci dat provádí metoda `deserialize` třídy `Serialization`.

Po rozbalení získáme seznam obsahující slovníky. Pro každou kombinaci argumentů aplikace „qpid_benchmark_mod.py“ jeden slovník, s klíčovými prvky:

- `cmd_options` – má přiřazen slovník, v němž jsou uloženy odpovídající vstupní argumenty aplikace „qpid_benchmark_mod.py“.
- `measured_values` – má přiřazen slovník s uloženými naměřenými hodnotami. Obsahuje klíčové prvky „send_tp“, „recv_tp“, „l_min“, „l_max“, „l_avg“, „tp_msg“, „tp_byte“. Každému z klíčů je přiřazena dvojice (<číslo měření>, <hodnota>).
- `summary` – slovník s vypočítanými hodnotami, obsahující klíčové prvky „send_tp“, „recv_tp“, „l_min“, „l_max“, „l_avg“, „tp_msg“, „tp_byte“. Každému z těchto klíčů je přiřazen slovník s klíčovými hodnotami „mean“, „mean_std_dev“, „valid_values“, „min_value“, „max_value“; každému prvku je pak přiřazena jeho vypočítaná hodnota.

Poslední slovník sdružuje souhrnné informace o měření.

Deserializaci a parsování těmito daty názorně ukazuje aplikace „compare_data.py“, která nejprve vypíše obsah zadaného souboru a v případě, že je zadaná dvojice souborů, vypíše obsahy obou souborů, najde v nich a vypíše porovnatelné hodnoty.

Aplikaci spustíme příkazem:

```
python compare_data.py -f <file1>[,<file2>] [option2] ...
```

Nápovědu můžeme zobrazit příkazem

```
python compare_data.py --help
```

Přehled argumentů a jejich význam:

- `-f`, `--file` – cesty k souborům se serializovanými daty oddělené čárkou; pokud je zadaná jen jedna cesta, program pouze vypíše obsah; povinný argument.
- `-o`, `--identical-options` – parametry příkazu „qpid_benchmark_mod.py“, které musí být stejné, aby hodnoty byly porovnatelné.
- `--print-data` – pokud je roven „True“, program vypíše kromě souhrnných vypočítaných dat a obecných informací o měření také celý obsah souboru.

Předpoklad je, že archivovaná data najdou využití hlavně při srovnávání výkonnosti jednotlivých verzí produktu nebo platforem (například klienti běžící na Windows s klienty na Linuxu apod.).

Ukládány jsou i původně naměřené hodnoty (včetně těch, které jsou později odstraněné jako odlehlé), aby bylo možné v budoucnu aplikovat i jiné metody zpracování výsledků.

ZÁVĚR

Výsledkem diplomové práce je aplikace „qpid_benchmark_stats“ implementovaná v programovacím jazyce Python. Aplikace spouští „qpid_benchmark_mod.py“, mírně upravenou verzi aplikace „qpid-cpp-benchmark“, používanou pro měření propustnosti (*throughput*) a časové odezvy (*latency*) Red Hat Enterprise MRG Messaging brokeru, přijme data z jejího standardního výstupu a zpracuje. Výsledkem je pětice hodnot pro každou sledovanou veličinu: střední hodnota, standardní směrodatná odchylka průměru, počet hodnot, se kterými bylo počítáno, minimální a maximální hodnota. Nakonec všechna data, včetně naměřených hodnot, uloží v serializované formě do souboru pro pozdější využití.

Aplikace je plně funkční a splňuje předem stanovené požadavky. Bude využívána testovacím týmem produktu Red Hat Enterprise MRG k přesnějšímu statistickému určení ukazatelů výkonnosti tohoto produktu.

Teoretická část práce čtenáře seznamuje s produktem Red Hat Enterprise Messaging a popisuje použité metody zpracování naměřených hodnot.

Praktická část obsahuje specifikaci, programátorskou dokumentaci a uživatelskou příručku.

V příloze P I je k nahlédnutí vygenerovaná dokumentace aplikace ze zdrojového kódu a v příloze P II ukázka výstupu aplikace „qpid_benchmark_stats.py“.

Na přiloženém disku ve složce DATA je uložen výstup aplikace „qpid_benchmark_stats.py“. Měření bylo prováděno na Red Hat Enterprise Messaging Brokeru verze 2.3 nainstalovaném na vícejádrovém serveru s operačním systémem Red Hat Enterprise Linux Server 6.4. Aplikace „qpid_benchmark_stats.py“, „qpid_benchmark_mod.py“ a klientské aplikace „qpid-send“ a „qpid-receive“ byly spouštěny na stejném počítači, pro potlačení vlivu kvality síťového připojení na výsledky měření.

Tímto diplomová práce splňuje zadané požadavky.

CONCLUSION

The result of the diploma thesis is the “qpid_benchmark_stats” application, implemented in Python programming language. The application runs the “qpid_benchmark_mod.py” tool, a slightly modified “qpid-cpp-benchmark” application, what is used for measuring of throughput and latency of the Red Hat Enterprise MRG Messaging broker; gets data from its standard output and processes them. The results are: mean, standard mean deviation, count of processed values, minimum and maximum value for every measurement. In the end the all data, including measured values, are serialized into a file for later usage.

The application is fully functional and meets the given requirements. It will be used by the Red Hat Enterprise MRG testing team to get more accurate statistical performance indicators of the product.

The teoretical part of the product tries to familiarize the reader with the Red Hat Enterprise MRG Messaging product and describes the used data processing methods.

The practical part contains the requirements specification, program documentation and user guide.

Attachment P I shows a generated documentation from the application source code and attachment P II contains an example of the “qpid_benchmark_stats.py” application output.

The attached disk contains the application output in the DATA directory. The measurements were performed on the Red Hat Enterprise Messaging Broker version 2.3, installed on a Red Hat Enterprise Linux Server 6.4 multi-core server machine. All applications, “qpid_benchmark_stats.py”, “qpid_benchmark_mod.py” and clients “qpid_send” and “qpid_receive” ran on the same machine as the broker to suppress the influence of network connection quality on the measured results.

The diploma thesis meets all defined requirements.

SEZNAM POUŽITÉ LITERATURY A INTERNETOVÉ ZDROJE

- [1] [Apache-SVN]/qpid/trunk/qpid/cpp/src/tests. *Apache-SVN* [online]. 2006 [cit. 2013-04-11]. Dostupné z:
<http://svn.apache.org/viewvc/qpid/trunk/qpid/cpp/src/tests/>
- [2] *The Apache Software Foundation* [online]. 2012 [cit. 2013-04-14]. Dostupné z:
<http://www.apache.org/>
- [3] Red Hat Enterprise MRG: Customer Portal [online]. 2013 [cit. 2013-01-18]. Dostupné z https://access.redhat.com/knowledge/docs/Red_Hat_Enterprise_MRG/
- [4] TŮMOVÁ, Olga. Metrologie a hodnocení procesů. 1. vyd. Praha: BEN - technická literatura, 2009, 231 s. ISBN 978-80-7300-249-7
- [5] ANDĚL, Jiří. Základy matematické statistiky. Vyd. 3. Praha: Matfyzpress, 2011, 358 s. ISBN 978-80-7378-162-0
- [6] BUDÍKOVÁ, Marie, Tomáš LERCH a Štěpán MIKOLÁŠ. Základní statistické metody. 1. vyd. Brno: Masarykova univerzita, 2005, viii, 170 s. ISBN 80-210-3886-1
- [7] PILGRIM, Mark. Ponorme se do Python(u) 3: Dive into Python 3. Praha: Cz.Nic, c2010, 430 s. CZ.NIC. ISBN 978-80-904248-2-1
- [8] *Red Hat* [online]. 2013 [cit. 2013-04-14]. Dostupné z: www.redhat.com

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACL	„Access Control List“, seznam jednotlivých oprávnění uživatelů.
AMQP	„Advanced Message Queuing Protocol“, otevřený standard pro posílání zpráv.
ANONYMOUS	Nejméně bezpečný ověřovací mechanismus SASL, nevyžaduje jméno ani heslo.
CPU	„Central Processing Unit“ (někdy též „procesor“), sekvenční obvod, který vykonává instrukce uložené v operační paměti
DIGEST-MD5	Ověřovací mechanismus SASL, bezpečný proti odposlouchávání sítě.
GSSAPI	Ověřovací mechanismus SASL, založený na síťovém autentizačním protokolu Kerberos.
ISO/OSI	Referenční model vypracovaný organizací ISO, řeší komunikaci v počítačových sítích.
MRG	„Red Hat Enterprise Messaging, Realtime, Grid“, produkt společnosti Red Hat.
PLAIN	Ověřovací mechanismus SASL, založený na jménu a heslu (bez šifrování).
RHN	„Red Hat Network“, systém služeb, zahrnující aktualizace softwaru, zákaznický servis apod., spravovaný firmou Red Hat, přístupný pro registrované uživatele.
SASL	„Simple Authentication and Security Layer“, obecná metoda pro autentizaci v klientů na serverech.
SSL	„Secure Sockets Layer“, protokol zajišťující zabezpečení komunikace po síti šifrováním a autentizací komunikujících uzlů.
SVN	„Subversion“, systém pro správu a verzování zdrojových kódů.
URL	„Uniform Resource Locator“, řetězec znaků, který jednoznačně specifikuje umístění subjektu na internetu.

XML „Extensible Markup Language“, vyvinutý a standardizovaný konsorciem W3C.

SEZNAM OBRÁZKŮ

<i>Obrázek 1: Komunikační model AMQP 0-10 [3]</i>	<i>14</i>
<i>Obrázek 2: Obecný model procesu měření [4]</i>	<i>19</i>
<i>Obrázek 3: Diagram tříd</i>	<i>36</i>
<i>Obrázek 4: Obsah balíčku „qpid_benchmark_stats.zip“</i>	<i>39</i>
<i>Obrázek 5: Obsah modulu „qpid_benchmark_stats_tools“</i>	<i>40</i>

SEZNAM PŘÍLOH

P I DOKUMENTACE VYGENEROVANÁ NÁSTROJEM PYDOC

P II UKÁZKA VÝSTUPU APLIKACE „QPID-BENCHMARK_STATS“

PŘÍLOHA P I: DOKUMENTACE VYGENEROVANÁ NÁSTROJEM PYDOC

Help on module `qpid_benchmark_stats`:

NAME

`qpid_benchmark_stats`

FILE

`/root/qpid_benchmark_stats/qpid_benchmark_stats.py`

DESCRIPTION

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
#
# =====
# Tool for statistical processing of data measured by qpid_benchmark
# =====
# Content:
# -----
# QpidBenchmarkStats - class provided the main functionality
# main() method declaration
#
# Processed data:
# -----
# send-tp - sender throughput [messages/s]
# recv-tp - receiver throughput [messages/s]
# l-max - maximum latency of a message [ms]
# l-min - minimum latency of a message [ms]
# l-avg - average latency of a message [ms]
# tp-msg - total throughput [messages/s]
# tp-byte - total throughput [bytes/s]
#
```

```

# Results:
# -----
# mean
# mean std dev - standard deviation of mean
# valid values - count of valid values after removing all measurements with
#                 extreme values
# min value - minimum measured value (without values in the removed
#                 measurements)
# max value - maximum measured value (without values in the removed
#                 measurements)
#
# Usage: ./qp_id_benchmark_stats.py [options]
# For more information run ./qp_id_benchmark_stats.py --help

```

CLASSES

QpidBenchmarkStats

```

class QpidBenchmarkStats
|   Instance of the class processes measured data by 'qp_id_benchmark_mod.py'.
|
|   Methods defined here:
|
|   check_int_val(self, in_val, min_val=None, max_val=None)
|       The method checks if in_val is between min_val and max_val;
|       returns True or False.
|       in_val - number to be checked
|       min_val - minimum value to compare
|       max_val - maximum value to compare
|
|   convert_items_to_bool(self, in_list)
|       The method converts all items from in_list to boolean type.
|       in_list - list of values to convert
|
|   convert_items_to_int(self, in_list, min_val=None, max_val=None)
|       The method converts all items from in_list to integer type
|       and checks range of the items.
|       in_list - list of values
|       min_val - minimum value to compare
|       max_val - maximum value to compare
|
|   create_list(self, opt_val)
|       The method returns a list created from the string argument.
|       opt_val - values separated by commas, what should be added into the list
|
|   fill_in_data_list(self, data_set, id, max_length)
|       The method fills a data_list by data with index id for statistical
|       processing and returns it. The list contains tuples
|       (measurement_number, value).

```

```

|     data_set - list of measured data got from qpid_benchmark_mod stdout
|     id - number of measurement
|     max_lenght - maximum length of the line with measured values (count of the
|                 variables)
|
|     get_datetime_string(self, dt)
|         The method gets the datetime data as a readable string.
|         dt - datetime value to convert
|
|     print_footer(self, b_info)
|         The method prints information about machine, time and description.
|         b_info - dictionary with basic information to be printed
|
|     print_results(self, tup_list)
|         The method prints results in readable format.
|         tup_list - list of tuples with results to print (<mean>, <mean std dev>,
|                 <valid values>, <min value>, <max value>);
|                 one tuple for every variable.
|
|     run(self)
|         Reads options, checks them and runs qpid_benchmark_mod; gets measured
|         values from its stdout, processes them and all data serializes into a file.

```

DATA

```
PIPE = -1
```

Help on module qpid_benchmark_stats_tools:

NAME

```
qpid_benchmark_stats_tools
```

FILE

```
/root/qpid_benchmark_stats/qpid_benchmark_stats_tools.py
```

DESCRIPTION

```

# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements. See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership. The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,

```

```

# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#
# =====
# Module with statistical calculations and other tools
# (needed by "qpid_benchmark_stats.py" tool)
# =====
# Content:
# -----
# StatsProcessing - class for basic statistical computing (mean, scattering,
#                  standard deviation, finding and removing extreme values)
# DataProcessing - class for statistical processing of measured data by
#                  "qpid_benchmark.py" tool
# Exception classes:
# - ExecutionException - occurs if a called application fails
# - InvalidDataException - occurs if an operation with given data cannot be done
# - InvalidValueException - occurs if any parameter has invalid value
# - CannotConvertException - occurs if any value cannot be converted into a
#                           required format.
# - InvalidPathException - occurs if given path does not exist
# DataStore - class for storage set of measured data into dictionaries
#             for serialization
# Serialization - class for adding basic data about machine and time into the
#                 data structure for serialization and serialization
#                 / deserialization

```

CLASSES

```

exceptions.Exception(exceptions.BaseException)
    CannotConvertException
    ExecutionException
    InvalidDataException
    InvalidPathException
    InvalidValueException
DataProcessing
DataStore
Serialization
StatsProcessing

class CannotConvertException(exceptions.Exception)
|   Exception class for failed conversions.
|
|   Method resolution order:
|       CannotConvertException
|       exceptions.Exception
|       exceptions.BaseException

```

```

|     __builtin__.object
|
| Data descriptors defined here:
|
|     __weakref__
|         list of weak references to the object (if defined)
|
| -----
| Methods inherited from exceptions.Exception:
|
|     __init__(...)
|         x.__init__(...) initializes x; see x.__class__.__doc__ for signature
|
| -----
| Data and other attributes inherited from exceptions.Exception:
|
|     __new__ = <built-in method __new__ of type object>
|         T.__new__(S, ...) -> a new object with type S, a subtype of T
|
| -----
| Methods inherited from exceptions.BaseException:
|
|     __delattr__(...)
|         x.__delattr__('name') <==> del x.name
|
|     __getattr__(...)
|         x.__getattr__('name') <==> x.name
|
|     __getitem__(...)
|         x.__getitem__(y) <==> x[y]
|
|     __getslice__(...)
|         x.__getslice__(i, j) <==> x[i:j]
|
|         Use of negative indices is not supported.
|
|     __reduce__(...)
|
|     __repr__(...)
|         x.__repr__() <==> repr(x)
|
|     __setattr__(...)
|         x.__setattr__('name', value) <==> x.name = value
|
|     __setstate__(...)
|
|     __str__(...)
|         x.__str__() <==> str(x)

```



```

|
|  __unicode__(...)
|
|  -----
|  Data descriptors inherited from exceptions.BaseException:
|
|  __dict__
|
|  args
|
|  message
|

class DataProcessing
|  Class for statistical processing of measured data.
|
|  Methods defined here:
|
|  __init__(self, data_lists)
|      data_lists - lists of tuples (<number>, <value>)
|
|  get_meas_numbers(self, it_id, values)
|      The method returns numbers of measurements which contain
|      given values of the specified variable.
|      it_id - id number of variable
|      values - values of it_id variable for which the
|               measurements should be found
|
|  get_stats(self)
|      The method processes the data; returns a list of tuples (mean,
|      standard deviation of mean, count of the valid values, minimum value,
|      maximum value) from lists of values.
|
|  get_values_to_process(self, i_list)
|      The method returns a list of values to process
|      (the second item of tuples).
|      i_list - list of tuples (<number>, <value>) with measured
|               values of one variable
|
|  remove_extreme_measurements(self, rm_ext)
|      The method removes measurements which contain extreme values;
|      rm_ext - maximum count of deleted measurements
|
|  remove_measurements(self, numbers)
|      The method removes measurements with given numbers.
|      numbers - numbers of measurements to be removed
|

class DataStore
|  Class contains methods to create data structures for serialization

```

```

| from one set of measurements.
|
| Methods defined here:
|
| __init__(self)
|
| add_cmd_options(self, cmd_opts)
|     The method adds command options to the self.cmd_options dictionary.
|     cmd_opts - list of command options
|
| add_measured_values(self, m_values)
|     The method adds measured values to the self.measured_values dictionary.
|     m_values - list of measurements
|
| add_summary(self, sum)
|     The method adds computed results to the self.summary dictionary.
|     sum - set of results from one set of measurements (list of tuples)
|
| get_data(self)
|
class ExecutionException(exceptions.Exception)
| Exception class for failures of application run.
|
| Method resolution order:
|     ExecutionException
|     exceptions.Exception
|     exceptions.BaseException
|     __builtin__.object
|
| Data descriptors defined here:
|
| __weakref__
|     list of weak references to the object (if defined)
|
| -----
| Methods inherited from exceptions.Exception:
|
| __init__(...)
|     x.__init__(...) initializes x; see x.__class__.__doc__ for signature
|
| -----
| Data and other attributes inherited from exceptions.Exception:
|
| __new__ = <built-in method __new__ of type object>
|     T.__new__(S, ...) -> a new object with type S, a subtype of T
|
| -----
| Methods inherited from exceptions.BaseException:

```

```

__delattr__(...)
    x.__delattr__('name') <==> del x.name

__getattr__(...)
    x.__getattr__('name') <==> x.name

__getitem__(...)
    x.__getitem__(y) <==> x[y]

__getslice__(...)
    x.__getslice__(i, j) <==> x[i:j]

    Use of negative indices is not supported.

__reduce__(...)

__repr__(...)
    x.__repr__() <==> repr(x)

__setattr__(...)
    x.__setattr__('name', value) <==> x.name = value

__setstate__(...)

__str__(...)
    x.__str__() <==> str(x)

__unicode__(...)

-----
Data descriptors inherited from exceptions.BaseException:

__dict__

args

message

class InvalidDataException(exceptions.Exception)
    Exception class for invalid input data.

    Method resolution order:
        InvalidDataException
        exceptions.Exception
        exceptions.BaseException
        __builtin__.object

```

```

| Data descriptors defined here:
|
| __weakref__
|     list of weak references to the object (if defined)
|
| -----
| Methods inherited from exceptions.Exception:
|
| __init__(...)
|     x.__init__(...) initializes x; see x.__class__.__doc__ for signature
|
| -----
| Data and other attributes inherited from exceptions.Exception:
|
| __new__ = <built-in method __new__ of type object>
|     T.__new__(S, ...) -> a new object with type S, a subtype of T
|
| -----
| Methods inherited from exceptions.BaseException:
|
| __delattr__(...)
|     x.__delattr__('name') <==> del x.name
|
| __getattr__(...)
|     x.__getattr__('name') <==> x.name
|
| __getitem__(...)
|     x.__getitem__(y) <==> x[y]
|
| __getslice__(...)
|     x.__getslice__(i, j) <==> x[i:j]
|
|     Use of negative indices is not supported.
|
| __reduce__(...)
|
| __repr__(...)
|     x.__repr__() <==> repr(x)
|
| __setattr__(...)
|     x.__setattr__('name', value) <==> x.name = value
|
| __setstate__(...)
|
| __str__(...)
|     x.__str__() <==> str(x)
|
| __unicode__(...)

```

```

|
| -----
| Data descriptors inherited from exceptions.BaseException:
|
|   __dict__
|
|   args
|
|   message
|
class InvalidPathException(exceptions.Exception)
|   Exception class for not existing paths.
|
|   Method resolution order:
|       InvalidPathException
|       exceptions.Exception
|       exceptions.BaseException
|       __builtin__.object
|
|   Data descriptors defined here:
|
|   __weakref__
|       list of weak references to the object (if defined)
|
| -----
| Methods inherited from exceptions.Exception:
|
|   __init__(...)
|       x.__init__(...) initializes x; see x.__class__.__doc__ for signature
|
| -----
| Data and other attributes inherited from exceptions.Exception:
|
|   __new__ = <built-in method __new__ of type object>
|       T.__new__(S, ...) -> a new object with type S, a subtype of T
|
| -----
| Methods inherited from exceptions.BaseException:
|
|   __delattr__(...)
|       x.__delattr__('name') <==> del x.name
|
|   __getattr__(...)
|       x.__getattr__('name') <==> x.name
|
|   __getitem__(...)
|       x.__getitem__(y) <==> x[y]
|

```

```

|  __getslice__(...)
|      x.__getslice__(i, j) <==> x[i:j]
|
|      Use of negative indices is not supported.
|
|  __reduce__(...)
|
|  __repr__(...)
|      x.__repr__() <==> repr(x)
|
|  __setattr__(...)
|      x.__setattr__('name', value) <==> x.name = value
|
|  __setstate__(...)
|
|  __str__(...)
|      x.__str__() <==> str(x)
|
|  __unicode__(...)
|
|  -----
|  Data descriptors inherited from exceptions.BaseException:
|
|  __dict__
|
|  args
|
|  message
|
|
|  class InvalidValueException(exceptions.Exception)
|      Exception class for invalid value.
|
|      Method resolution order:
|          InvalidValueException
|          exceptions.Exception
|          exceptions.BaseException
|          __builtin__.object
|
|      Data descriptors defined here:
|
|      __weakref__
|          list of weak references to the object (if defined)
|
|      -----
|      Methods inherited from exceptions.Exception:
|
|      __init__(...)
|          x.__init__(...) initializes x; see x.__class__.__doc__ for signature

```

```

|
| -----
| Data and other attributes inherited from exceptions.Exception:
|
| __new__ = <built-in method __new__ of type object>
|     T.__new__(S, ...) -> a new object with type S, a subtype of T
|
| -----
| Methods inherited from exceptions.BaseException:
|
| __delattr__(...)
|     x.__delattr__('name') <==> del x.name
|
| __getattr__(...)
|     x.__getattr__('name') <==> x.name
|
| __getitem__(...)
|     x.__getitem__(y) <==> x[y]
|
| __getslice__(...)
|     x.__getslice__(i, j) <==> x[i:j]
|
|     Use of negative indices is not supported.
|
| __reduce__(...)
|
| __repr__(...)
|     x.__repr__() <==> repr(x)
|
| __setattr__(...)
|     x.__setattr__('name', value) <==> x.name = value
|
| __setstate__(...)
|
| __str__(...)
|     x.__str__() <==> str(x)
|
| __unicode__(...)
|
| -----
| Data descriptors inherited from exceptions.BaseException:
|
| __dict__
|
| args
|
| message

```

```

class Serialization
|   Class for collection, serialization and deserialization data.
|
|   Methods defined here:
|
|   __init__(self)
|
|   add_basic_info(self, description=None)
|       The method adds information about machine, time and description
|       as the first dictionary into the data_container.
|       description - string to be added into the measurement
|
|   add_data(self, data_store)
|       The method adds a set of data into the self.data_container.
|       data_store - dictionary with data (measured values, command
|                   options and results)
|
|   deserialize(self, store_file)
|       The method deserializes data in the "store_file".
|       store_file - path to the file with the serialized data
|
|   get_cpu_count(self)
|       The method gets the number of CPUs on a system.
|
|   get_data(self)
|       The method gets "self.data_container" value
|
|   serialize(self, store_file, store_dir)
|       The method serializes self.data_container into
|       the "store_dir/store_file".
|       store_file - name of file where data should be serialized to
|       store_dir - path to the store_file
|
class StatsProcessing
|   Class for statistical computing.
|
|   Methods defined here:
|
|   __init__(self, values)
|       Fills in a list of data for processing.
|       values - data to process
|
|   compute_mean(self)
|       The method returns an arithmetic mean from values.
|
|   compute_mean_std_dev(self)
|       The method returns a sample standard deviation of the arithmetic mean.
|

```



```

| compute_scattering(self)
|     The method returns a sample scattering from values.
|
| compute_standard_deviation(self)
|     The method returns a sample standard deviation of one measurement.
|
| find_extreme_values(self, max_extremes)
|     The method finds extreme values, removes them
|     from the self.values list and returns them.
|     max_extremes - count of extremes what can be removed
|
| get_maximum(self)
|     The method returns a maximum value.
|
| get_minimum(self)
|     The method returns a minimum value.
|
| get_values(self)
|     The method gets processed values.
|
| is_extreme(self, value)
|     The method tests if the value is extreme and returns the result.
|     value - checked value
|
| remove_negs(self)
|     The method removes values less than 0 from the self.values.

```

Help on module `qpid_benchmark_mod`:

NAME

`qpid_benchmark_mod`

FILE

`/root/qpid_benchmark_stats/qpid_benchmark_mod.py`

DESCRIPTION

```

# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements. See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership. The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#

```

```

# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#

```

CLASSES

```

Clients
ReadyReceiver
RoundRobin
subprocess.Popen(__builtin__.object)
    PopenCommand

```

```
class Clients
```

```

|  Methods defined here:
|
|  __init__(self)
|
|  add(self, client)
|
|  kill(self)

```

```
class PopenCommand(subprocess.Popen)
```

```

|  Like Popen but you can query for the command
|
|  Method resolution order:
|      PopenCommand
|      subprocess.Popen
|      __builtin__.object
|

```

```

|  Methods defined here:
|
|  __init__(self, command, *args, **kwargs)
|

```

```

|  -----
|  Methods inherited from subprocess.Popen:
|

```

```

|  __del__(self, _maxint=9223372036854775807, _active=[])
|

```

```

|  communicate(self, input=None, timeout=None)
|
|      Interact with process: Send data to stdin. Read data from
|      stdout and stderr, until end-of-file is reached. Wait for
|      process to terminate. The optional input argument should be
|      a string to be sent to the child process, or None, if no data
|      should be sent to the child.
|

```

```

|     communicate() returns a tuple (stdout, stderr).
|
|     The optional "timeout" argument is a non-standard addition to
|     Python 2.6. If supplied, it is a number of seconds, which can be an
|     integer or a float (though there are no precision guarantees). A
|     TimeoutExpired exception will be raised after the given number of
|     seconds elapses, if the call has not yet returned.
|
|     kill(self)
|         Kill the process with SIGKILL
|
|     poll(self)
|
|     send_signal(self, sig)
|         Send a signal to the process
|
|     terminate(self)
|         Terminate the process with SIGTERM
|
|     wait(self, timeout=None, endtime=None)
|         Wait for child process to terminate. Returns returncode
|         attribute.
|
|         The optional "timeout" argument is a non-standard addition to
|         Python 2.6. If supplied, it is a number of seconds, which can be
|         an integer or a float (though there are no precision guarantees).
|         A TimeoutExpired exception will be raised after the given number of
|         seconds elapses, if the call has not yet returned.
|
|     -----
|     Data descriptors inherited from subprocess.Popen:
|
|     __dict__
|         dictionary for instance variables (if defined)
|
|     __weakref__
|         list of weak references to the object (if defined)
|
class ReadyReceiver
|     A receiver for ready messages
|
|     Methods defined here:
|
|     __init__(self, queue, broker)
|
|     wait(self, receivers)

class RoundRobin

```

```

|   Methods defined here:
|
|   __init__(self, items)
|
|   next(self)

```

FUNCTIONS

```

error_msg(out, err)

first_line(p)

flatten(l)

get_queues_status(brks, queue_names)
    The method gets count of enqueued / dequeued messages / bytes
    brks - list of brokers
    queue_names - list of queue names

main()

parse(parser, lines)

parse_receivers(receivers)

parse_senders(senders)

posix_quote(string)
    Quote a string for use as an argument in a posix shell

print_data(send_stats, recv_stats, total_tp_msg, total_tp_bytes)

print_header(timestamp)

print_summary(send_stats, recv_stats, total_tp_msg, total_tp_bytes)

recreate_queues(queues, brokers, no_delete, opts)

ssh_command(host, command)
    Convert command into an ssh command on host with quoting

start_receive(queue, index, opts, ready_queue, broker, host)

start_send(queue, opts, broker, host)

```

DATA

```

PIPE = -1
STDOUT = -2
clients = <qpid_benchmark_mod.Clients instance>

```

```
op = <optparse.OptionParser instance>
single_quote_re = <_sre.SRE_Pattern object>
```

Help on module compare_data:

NAME

compare_data

FILE

/root/qpid_benchmark_stats/compare_data.py

DESCRIPTION

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
#
# =====
# Example how to read and compare two *.data files
# =====
# Usage: ./compare_data.py [--print-data]
#         -f <file_name1>[,<file_name2>] [-o <list_of_opts>]
# For more information run ./compare_data.py --help
```

FUNCTIONS

```
get_comparable_data(data1, data2, i_opts)
    The method gets list of lists containing comparable data
    (data got from measurements from commands with the same options).
    data1, data2 - deserialized data (lists) from 1st and 2nd files
    i_opts - list of keys (option names) which should be identical

get_datetime_string(dt)
    The method gets the datetime data as a readable string.
    dt - datetime value to convert
```

```

is_comparable(dict1, dict2, i_opts)
    The method compares command options in dict1 and dict1 dictionaries
    and returns result (True if they are the same, in another case False);
    dict1, dict2 - dictionaries with data from one command
    i_opts - list of keys (option names) which should be identical
            in the both dictionaries.

main()
    Reads options and checks them; deserializes data from files,
    prints them; finds the comparable data according to options and
    prints them.

print_basic_info(data)
    The method prints basic information part from the data list.
    data - deserialized data (list)

print_command(i_dict)
    The method prints the executed command.
    i_dict - dictionary with data

print_content_of_data_file(data, print_all=False)
    The method prints data contained in the "*.data" file.
    data - deserialized data (list)

print_results(c_data)
    The method prints comparable results.
    c_data - list of lists of comparable data

print_summary(i_dict)
    The method prints the summary of one executed command.
    i_dict - dictionary with data

```

PŘÍLOHA P II: UKÁZKA VÝSTUPU APLIKACE „QPID-BENCHMARK_STATS“

```
./qpid_benchmark_stats.py -m 100 --content-size 1024 -q 1 -s 1 -r 1 --description "MRG Messaging version 2.3 II"
```

Running command:

```
python qpid_benchmark_mod.py -b 127.0.0.1 -q 1 -s 1 -r 1 -m 100 --qpid-send-path=/opt/rh-qpid/clients/qpid-send --qpid-receive-path=/opt/rh-qpid/clients/qpid-receive --queue-name benchmark --content-size 1024 --ack-frequency 100 --repeat 55 --durable False --fill-drain False
```

(First 5 measurements will be removed.)

Summary:

	send-tp	recv-tp	l-min	l-max	l-avg	tp-msg	tp-byte
	[msg/s]	[msg/s]	[ms]	[ms]	[ms]	[msg/s]	[bytes/s]
mean	32590	43570	135.73	137.11	136.35	314	321689
mean std dev	1433	1290	3.21	3.3	3.25	5	4860
valid values	50	50	50	50	50	50	50
min value	20546	38632	91.13	91.27	91.2	254	260134
max value	50092	69081	198.45	200.62	199.43	382	391247

Running command:

```
python qpid_benchmark_mod.py -b 127.0.0.1 -q 1 -s 1 -r 1 -m 100 --qpid-send-path=/opt/rh-qpid/clients/qpid-send --qpid-receive-path=/opt/rh-qpid/clients/qpid-receive --queue-name benchmark --content-size 1024 --ack-frequency 100 --repeat 55 --durable True --fill-drain False
```

(First 5 measurements will be removed.)

Summary:

	send-tp	recv-tp	l-min	l-max	l-avg	tp-msg	tp-byte
	[msg/s]	[msg/s]	[ms]	[ms]	[ms]	[msg/s]	[bytes/s]
mean	35440	43050	134.68	135.87	135.21	316	324334
mean std dev	1619	1092	3.77	3.88	3.82	5	5375
valid values	48	48	48	48	48	48	48
min value	21408	38193	90.61	90.8	90.7	257	263831
max value	48852	70027	195.52	197.66	196.49	373	382227

=====

Done.

Measurement description:

MRG Messaging version 2.3 II

Machine info:

```
{'uname': ('Linux', 'mrg-qe-06.lab.eng.brq.redhat.com', '2.6.32-358.2.1.el6.x86_64', '#1 SMP Wed Feb 20 12:17:37 EST 2013', 'x86_64', 'x86_64'), 'cpus': 8}
```

Start time of measurement: 2013-4-14 13:10:40

End time of measurement: 2013-4-14 13:17:1

Time duration: 0:06:20.525629