TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan osasto

JAAKKO LUOTO

TIETOKANTAKLUSTERIN TEHOKKUUS- JA SAATAVUUSTESTAUS

Diplomityö

Aihe hyväksytty osastoneuvoston kokouksessa
8.12.2004

Tarkastajat: Prof. Tommi Mikkonen (TTY)
Tero Laine (Nokia, Networks)

Alkulause

Tämä diplomityö on tehty Tampereen teknillisen yliopiston Ohjelmistotekniikan laitoksella. Työn pohjana olevan tutkimuksen alullepanijana toimi Nokia Networks.

Haluan esittää kiitokseni kaikille, jotka auttoivat työn valmistumisessa. Erityisesti haluan kiittää professori Tommi Mikkosta kommenteista ja opastuksesta diplomityön kirjoittamisessa, sekä Nokia Networksin Tero Lainetta hyvistä ideoista ja kannustuksesta projektin toteutusvaiheessa. Kiitän järjestelmäsuunnittelija Timo Alataloa yhteistyöstä testiympäristöä rakennettaessa ja tutkija Marko Leppästä avusta työn kieliasun viimeistelyssä. Kiitokset myös kollegoilleni Jerome Rannikolle ja Niko Kummulle, joiden LaTeX-asettelua käytin tämän dokumentin pohjana.

Ohjelmistotekniikan laitosta kiitän piristävästä ja joustavasta työilmapiiristä, joiden ansioista työn valmistuminen onnistui muiden työtehtävien ohessa.

Lisäksi suuret kiitokset kotiväelle kannustuksesta ja henkisestä tuesta.

Tampereella 26.4.2006

Jaakko Luoto

Abstract

TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Information Technology

Institute of Software Systems

LUOTO, JAAKKO: Performance and Availability Benchmark of a Database Cluster

Master of Science Thesis, 52 pages.

Examiners: Prof. Tommi Mikkonen (TUT, Institute of Software Systems) and
Tero Laine (Nokia, Networks)

Funding: Nokia, Networks

May 2006

Keywords: Database Clustering, Performance Benchmark, High Availability

Utilization rate of network services is on a constant raise. Shorter response times and reliability are expected. This sets high availability and performance requirements to the storage systems of services. The challenge can be answered to by distributing the data store on a server cluster, with the ability to scale the performance level by adding more servers to the group. This also makes it possible to have the other computers continue to provide services when one component fails. Some free or low-cost software is available with which distribution can be implemented using commodity hardware. It is not simple, however, to evaluate the quality of a cluster, because the criteria is dependent on the requirements of the application.

The aim of this work was at building a testing environment using selected clustering solutions based on the free MySQL database management system, and at assessing their applicability as the basis of a Home Location Register, a database used in mobile communication networks. TM1 benchmarking software with different combinations of test parameters is used to evaluate the performance and availability of the environment.

We conclude that improvement is still needed on the tested services to make them useful in the HLR field. It is also noted that the test results depend on many variables and that the test method behaves unexpectedly, which is why the gathered results alone can not be used to draw conclusions about the systems' performance in any other environment than the one used in this work.

Tiivistelmä

TAMPEREEN TEKNILLINEN YLIOPISTO

Tietotekniikan osasto

Ohjelmistotekniikka

LUOTO, JAAKKO: Tietokantaklusterin tehokkuus- ja saatavuustestaus

Diplomityö, 52 s.

Tarkastajat: prof. Tommi Mikkonen (TTY, Ohjelmistotekniikka) ja
Tero Laine (Nokia, Networks)

Rahoitus: Nokia, Networks

Toukokuu 2006

Avainsanat: tietokantaklusterointi, tehokkuustestaus, korkea saatavuus

Tietoliikennepalvelujen käyttöaste kasvaa jatkuvasti. Niiltä odotetaan yhä lyhyempiä vasteaikoja ja häiriötöntä toimintaa. Tämä asettaa korkeat vaatimukset palvelujen tietovarastojen suorituskyky- ja saatavuusominaisuuksille. Haasteeseen vastataan hajauttamalla tietosisältö palvelimista muodostettuun klusteriin, jonka suorituskyky on skaalattavissa palvelimien määrää kasvattamalla. Tällöin yksittäisen palvelimen vioittuminen ei myöskään pysäytä koko järjestelmän toimintaa. Nykyisin on saatavilla ilmaisia ja edullisia ohjelmistoja, joiden avulla hajautus voidaan toteuttaa kuluttajahintaisilla laitteilla. Klusterin käyttökelpoisuuden arviointi ei kuitenkaan ole yksiselitteistä, sillä mittauskriteerit vaihtelevat eri sovellusten tarpeiden mukaisesti.

Tässä diplomityössä valittiin joukko ilmaiseen MySQL-tietokantajärjestelmään perustuvia klusterointiratkaisuja ja rakennettiin testiympäristö, jossa voitiin arvioida ohjelmistojen käyttökelpoisuutta telekommunikaatioverkoissa käytettävän HLR-tietokannan alustana. Ympäristön tehokkuutta ja saatavuutta arvioitiin hyödyntäen TM1-testiohjelmaa käyden läpi erilaisia yhdistelmiä valituista testiparametreista.

Hajautusohjelmistojen ominaisuuksien huomattiin tarvitsevan vielä kehittämistä, jotta palveluja voitaisiin hyödyntää HLR-käytössä. Lisäksi testiympäristössä havaittiin muuttujia olevan liikaa ja testimenetelmän käyttäytyvän liian ennalta-arvaamattomasti, jotta niistä voitaisiin tehdä perusteltuja päätelmiä suorituskyvystä muissa kuin tässä yksittäisessä ympäristössä.

Sisällysluettelo

Lyhenteitä ja selityksiä

AQTh	Average Qualified Throughput. Keskimääräinen MQTh.
COMMIT	SQL:ssä transaktion lopettava lause, joka kirjaa edeltävien lauseiden tekemät muutokset pysyvästi tietokantaan.
GPL	GNU General Public License. Vapaa ohjelmistolisenssi.
HLR	Home Location Register. Tietokanta mobiiliverkon tilaajatiedoille.
IP	Internet Protocol. Tietoliikenneprotokolla, jota käyttävillä verkkolaitteilla on yksilöivä IP-osoite.
Klusteri	Joukko tietokoneita, joilla on yhteinen tietovarasto ja jotka toimivat jonkin yhteisen tehtävän suorittamiseksi.
MQTh	Maximum Qualified Throughput. Tietokantaan tehtyjen onnistuneiden transaktioiden lukumäärä sekunnissa.
NTTM1	Non-transactional TM1. TM1-testiohjelmasta muokattu ei-transaktiollinen versio.
NDB	Network Database. MySQL Clusterin tietokantamoottori.
ODBC	Open Database Connectivity. Avoin ohjelmointirajapinta tietokannan käsittelyyn.
ROLLBACK	SQL:ssä transaktion lopettava lause, joka peruuttaa edeltävien lauseiden tekemien muutosten kirjaamisen tietokantaan.
SQL	Structured Query Language. Kyselykieli tietokantajärjestelmien käyttämiseen.
TCP	Transmission Control Protocol. IP:n päällä toimiva yhteydellinen tietoliikenneprotokolla.
Testikokoonpano	Yhdistelmä seuraavista testiparametrien arvoista: tietokanta-alusta, solmujen lukumäärä ja populaation koko.
Testikonfiguraatio	Yhdistelmä seuraavista testiparametrien arvoista: asiakasprosessien lukumäärä, luku- ja kirjoitusoperaatioiden suhde sekä kaikki testikokoonpanossa määrätyt parametrit.
TKHJ	Tietokannanhallintajärjestelmä.
UDP	User Datagram Protocol. IP:n päällä toimiva yhteydetön tietoliikenneprotokolla.
TM1	Telecom One Benchmark. Testiohjelma HLR-tietokannan suorituskyvyn mittaamiseen.

Luku 1

Johdanto

Internetin ja muiden tietoliikennepalvelujen asiakasmäärät ovat jatkuvassa nousussa. Palvelut monimutkaistuvat, ja asiakkaiden ja palvelujen välillä kulkevan tiedon määrä kasvaa. Samaan aikaan liikennöintinopeudet nousevat, ja palveluilta odotetaan yhä lyhyempiä vasteaikoja ja häiriötöntä toimintaa. Tämä asettaa ankarat suorituskyky- ja saatavuusvaatimukset palveluiden tietovarastoille. Koska käyttöasteen odotetaan kasvavan jatkossakin, myös skaalautuvuus on tärkeä kriteeri tietokantaratkaisua valittaessa.

Verkkopalveluiden yleistyminen ja niiden koon kasvu on lisännyt edullisten palvelinratkaisujen tarvetta. Verkkopalvelujen katkeamaton toiminta on monille yrityksille nykyisin niin tärkeää, että esimerkiksi WWW-palvelun ja tietokannan asettaminen samalle yksittäiselle tietokoneelle aiheuttaa liiketoiminnalle tarpeettoman suuren riskin. Useat ilmaiset tai edulliset ohjelmistot on todettu riittävän luotettaviksi ratkaisuiksi ja otettu käyttöön palvelimissa. Yleisen yhdistelmän muodostavat Linux ja avoimeen lähdekoodiin perustuvat WWW- ja tietokantapalvelinohjelmistot, kuten Apache ja MySQL.

Ilmaisten ohjelmistojen saatavuusominaisuuksille asetetaan yhä enemmän painoarvoa ja palvelinohjelmistoissa on kasvava kysyntä kaupallisista palvelinratkaisuista tutuille ominaisuuksille. Esimerkiksi siirtyminen yhden koneen palvelusta monen koneen hajautettuun järjestelmään helpottuu, kun hajautuksen pohjalla voidaan käyttää vanhaa tietokantajärjestelmää. Tällöin vanhan pohjan päälle rakennettuja sovelluksia ei välttämättä tarvitse muuttaa lainkaan.

Monet ohjelmistot pyrkivät täyttämään tämän markkinaraon. Internetissä on saatavilla Linuxiin ja eri tietokannanhallintajärjestelmiin pohjautuvia ohjelmistotuotteita, joi-

den luvataan mahdollistavan korkean saatavuuden palveluita kuluttajahintaisista laitteista kootussa järjestelmässä, jossa joukko yhteisen tietosisällön jakavia tietokoneita muodostavat niin sanotun klusterin. Yleensä tuotteet pyrkivät tukemaan samanlaisia ominaisuuksia kuin alkuperäinenkin järjestelmä. Tietokantajärjestelmissä merkittävä ominaisuus voi olla esimerkiksi se, kuinka kattava osa SQL-kielen ominaisuuksista on tuettu.

Julkisia suorituskykymittauksia avoimen lähdekoodin tietokantajärjestelmien hajauttamisesta on saatavilla hyvin vähän. Omien suorituskykytestien tekemistä helpottaa kuitenkin se, että testaamiseen ei tarvita kalliita laitteita vaan muuttujien vaikutusta tuloksiin voidaan arvioida myös perus-PC:istä rakennetussa pienimuotoisessa klusterissa.

Tässä työssä haluttiin muodostaa hyvä yleiskuva tällä hetkellä saatavilla olevista ilmaisista tai edullisista klusteritoteutuksista ja vertailla niiden käyttökelpoisuutta tietokantapohjaisessa telekommunikaatiosovelluksessa ominaisuuksiensa ja tehokkuutensa perusteella. Tätä varten määriteltiin tietokantapalvelua mallintava laitteisto- ja ohjelmistoympäristö, jossa ajettiin joukko tietokantajärjestelmien suorituskykyä ja saatavuutta mittaavia testejä.

Työn rakenne on seuraava. Toisessa luvussa esitetään saatavuuden kannalta tärkeitä rinnakkaisten järjestelmien ominaisuuksia keskittyen erityisesti klusteroituihin tietokantapalveluihin sekä tarkastellaan lyhyesti keinoja niiden suorituskyvyn mittaamiseen. Luvussa kolme esitellään erilaisia arkkitehtuureja tietokantapalvelun hajauttamiseen ja ohjelmistoja, joilla sen toteutus on mahdollista. Neljännessä luvussa kuvataan niistä rakennettu testiympäristö, sen testikonfiguraatiot sekä käytetty testiohjelmisto ja siihen tehdyt muutokset. Testien tulokset esitetään ja niitä analysoidaan luvussa viisi. Luvussa kuusi kootaan yhteenveto tehdystä työstä ja sen tuloksista.

Luku 2

Rinnakkaiset palvelinjärjestelmät

2.1 Yleiskuvaus

Nykyaikaisessa tietokannassa käsitellään jaettua tietoa, jolla on monta yhtäaikaista käyttäjää. Rinnakkaisten asiakkaiden määrän mukana kasvaa myös tarvittava prosessorikuorma. Lopulta tulee vastaan tilanne, jossa kuormaa jakamassa on oltava useita rinnakkaisia prosessoreita, jotta tietokantaa käsittelevien operaatioiden vasteajat pysyisivät riittävän alhaisina. Erilaiset rinnakkaisia prosessoreita sisältävät arkkitehtuurit jaotellaan kolmeen ryhmään [25]:

- Yhteinen muisti (*shared-memory*). Prosessorit käyttävät samaa fyysistä muistia yhden tai useamman yhteisen muistiväylän kautta. Myös massamuisti on kaikkien prosessorien yhteiskäytössä.
- Yhteinen massamuisti (*shared-disk*). Prosessoreilla on oma keskusmuistinsa, mutta ne käyttävät yhteistä massamuistia, joka yleensä on kiintolevyjärjestelmä. Sen sisällön korruptoituminen vältetään lukitusten avulla.
- Yksityinen muisti (*shared-nothing*). Kullakin prosessorilla on oma muistinsa ja levynsä. Laitteet kommunikoivat keskenään vain verkkoliitynnän kautta.

Yhteiseen muistiin ja massamuistiin perustuvat ratkaisut vaativat perinteisesti juuri kyseiseen käyttötarkoitukseen suunniteltua laitteistoa. Esimerkiksi jaetulle levylle tallennettava tietokanta voidaan toteuttaa jaetun SCSI-väylän avulla, jonka skaalautuvuus on rajallinen. Tällainen keskitetty väylä, johon kytketään useita komponentteja,

on monimutkainen ja sitoo tietyn laitteistovalmistajan tuotteisiin järjestelmää laajennettaessa ja huollettaessa. Yhteismuistiarkkitehtuureja noudattavia laitteita on saatavissa lähinnä kalliina High End -tuotteina, jos tarvitaan yksittäistä moniprosessori-PC:tä nopeampi alusta.

Yksityisen muistin arkkitehtuuri on noussut suosituimmaksi rinnakkaisen tietokannan toteutustavaksi. Se on mainituista vaihtoehdoista edullisin, koska järjestelmä voidaan rakentaa täysin kuluttajahintaisia laitteita käyttäen. Laitteiden ei tarvitse olla lainkaan rinnakkaiseen käyttöön suunniteltuja, sillä rinnakkaisuus on toteutettu sovellustasolla. Laitekanta voi olla heterogeenistä, ja toteutuksesta riippuen jopa jokainen laite voi olla erilaiseen laitearkkitehtuuriin pohjautuva.

Joukkoa tietokoneita, joilla on yhteinen tietovarasto ja jotka toimivat jonkin yhteisen tehtävän suorittamiseksi, kutsutaan nimellä *klusteri* (*ryväs*, engl. *cluster*). Klusteriin liitettyjä toimijoita kutsutaan *solmuiksi* (engl. *node*). Tässä työssä solmu merkitsee fyysistä tietokonetta, ellei toisin mainita[1].

Klusterin solmut toimivat vikatilanteissa toistensa varasolmuina ja jakavat kuormaa keskenään. Klusterin rinnakkaisuus piilotetaan sen palveluja käyttävältä asiakkaalta siten, että se näyttää asiakkaalle vain yhdeltä tietokantahakuja ja -päivityksiä vastaanottavalta koneelta. Tällaisen yhdeksi koneeksi naamioitumisen (engl. *single system image*) etuna on se, että rinnakkaisuusjärjestelyjä voidaan muuttaa (esimerkiksi lisätä solmuja klusteriin) ilman, että se vaikuttaa asiakaspään toimintaan.

Edullisia laitteita ja ohjelmistoja hyödyntäviä klustereita kutsutaan Beowulf-klustereiksi. Nimi on peräisin NASA:n rahoittamasta vuonna 1994 aloitetusta rinnakkaista tietojenkäsittelyä tutkivasta Beowulf-projektista, jonka tutkijat rakensivat ensimmäisen Beowulf-klusterin kuluttajahintaisista PC-laitteista [14]. Beowulf-klusterit jaetaan kahteen luokkaan:

- Luokan 1 klusteri koostuu kokonaan standardista kuluttajien saatavissa olevasta tekniikasta (SCSI, Ethernet jne.)
- Luokan 2 klusterin toteutuksessa on mukana epästandardeja osia, kuten yhteen laitteistovalmistajaan sidotut (engl. *proprietary*) tuotteet.

[1]Nimeämiskäytäntö ei ole eri lähdemateriaaleissa yhtenäinen, sillä joskus solmulla tarkoitetaan klusteriin liitettyä fyysistä tietokonetta (tai prosessoria) ja joskus taas klusterin toimintaan osallistuvaa prosessia, joita yhdessä tietokoneessa voi olla käynnissä useita.

Luokassa 2 saavutetaan usein luokkaa 1 tehokkaammat klusterit, mutta samalla saatetaan menettää osa luokan 1 eduista: komponenttien edullinen hinta, ajurituki avoimen lähdekoodin yhteisöiltä ja riippumattomuus yksittäisestä valmistajasta.

2.2 Korkea saatavuus

Palvelujen *saatavuudella* (engl. *availability*) tarkoitetaan palvelun toiminta-ajan ja toimimattomuuteen kuluvan ajan suhdetta toisiinsa. Tämän suhteen maksimoiminen eli mahdollisimman *korkea saatavuus* (engl. *high availability*) on aina tärkeä tavoite hajautettuja palveluja kehitettäessä.

Internet-pohjainen palvelu ei koskaan kykene tarjoamaan täydellistä saatavuutta. Fyysiset viat, kuten kaapeleihin kompastelu ja laitteiden lämpöongelmat, voidaan poistaa lähes kokonaan riittävän valvonnan avulla. Perinteisistä laitteisto-ongelmista kuten verkkoyhteyksien ja tietokonekomponenttien vioista tai sähkökatkoksista aiheutuvista vikatilanteista voidaan selvitä kahdennus- ja varajärjestelmien avulla. On tärkeää tunnistaa ja kahdentaa kaikki yksittäiset järjestelmän osat, joiden rikkoutuminen tekisi koko järjestelmän toimintakyvyttömäksi (engl. *single point of failure*). Käytettävyydelle haasteita asettavat lisäksi mm. käyttöasteen kasvu, palvelujen monipuolistuminen ja ohjelmistojen päivitys- ja optimointitarpeet. Näitä haasteita kohdataan useimpien verkkopalveluiden ylläpidossa ja niiden vuoksi palvelun toiminta voidaan joutua ajoittain katkaisemaan omatoimisesti.

Mahdollisimman korkean saatavuuden takaamiseksi katkoihin ja niistä toipumiseen kuluva aika on minimoitava. Huoltoa on voitava tehdä yhdelle järjestelmän komponentille kerrallaan muiden hoitaessa sillä aikaa palvelua riittävällä palvelutasolla. Hajautus voidaan myös laajentaa maantieteellisesti useisiin toimipisteisiin. Sen avulla vähennetään paikallisten ongelmatilanteiden kuten onnettomuuksien, sähkökatkojen ja luonnonkatastrofien vaikutuksia palvelun toimintaan ja mahdollistetaan riittävän nopeat yhteydet palveluun kaikkialta palvelun peittoalueelta.

Palvelun saatavuutta mitattaan laskemalla jatkuvaa *toiminta-aikaa* (engl. *uptime*) eli sitä, kuinka suuren osan mitattavasta ajanjaksosta se on häiriöttömässä toimintakunnossa [2]. Toiminta-ajan vastakohtaa kutsutaan *seisokkiajaksi* (engl. *downtime*). Saatavuudelle voidaan laskea prosenttiluku suhteuttamalla keskimääräinen virhetilanteiden välillä kuluva aika ja keskimääräinen vian korjaamiseen kuluva aika:

$$saatavuusprosentti = \left(1 - \frac{korjausaika}{virheiden\ välinen\ aika}\right) \times 100.$$

Ympäri vuoden 24 tuntia päivässä toiminnassa olevassa palvelussa 99 prosentin saatavuus tarkoittaisi sitä, että palvelu on seisokissa 3,65 vuorokautta vuodessa. Tällainen saatavuus on useimmille kriittisille sovelluksille täysin riittämätön. Puhutaan yhdeksikköjen lukumääristä; Yleisesti tavoiteltava raja-arvo on 99,999-prosenttinen saatavuus eli ”viisi yhdeksikköä”. Se sallii korkeintaan 5,26 minuuttia seisokkiaikaa vuodessa.

Saatavuusprosentin laskukaava ei ota huomioon vakavuudeltaan erilaisia vikatilanteita, vaan olettaa palvelun olevan joko toiminnassa tai seisokissa. Yksittäinen vika ei kuitenkaan aina häiritse palvelun toimintaa niin paljoa, että asiakkaiden saama palvelun taso heikentyisi havaittavasti. Jos pientä vikaa ei haluta laskea seisokiksi, voidaan määritellä minimikriteerit riittävän hyvälle toiminnalle. Tietokantapalvelussa nämä kriteerit voisivat olla esimerkiksi tietokannan hakuoperaatioiden keskimääräinen kesto ja epäonnistuneiden transaktioiden lukumäärä aikayksikköä kohden. Usean koneen klusterissa yhden solmun toiminnan keskeytyminen ei vielä aiheuttaisi niin suurta tehohukkaa, että saatavuus laskisi raja-arvojen alapuolelle, mutta vastaava tilanne samanaikaisesti riittävän monessa solmussa laskettaisiin jo seisokiksi.

2.3 Replikointi

Yksityisen muistin arkkitehtuurin heikko kohta on helposti nähtävissä: kukin solmu omistaa oman muistinsa ja levynsä ja toimii palvelimena niiden sisältämälle tiedolle. Solmun vikaantuessa sen tiedot jäävät saavuttamattomiin, mikä voi aiheuttaa suuria ja aikaavieviä siirto-operaatioita muille solmuille. Tämän välttämiseksi osa klusterin kapasiteetista käytetään *replikointiin* (*toisinnus*) siten, että jokaisesta tietoalkiosta ylläpidetään kopiota eli *replikaa* useammassa solmussa.

Replikointitekniikoita luokitellaan sen mukaan, kuinka tietokannan muutokset levitetään kaikkiin replikoihin [9]: *ahkeraksi replikoinniksi* (engl. *eager replication*) kutsutaan sitä, että tietokantaan tehtävä muutos hyväksytetään kaikilla replikoilla ennen kuin se astuu voimaan. Tällöin tietokannan päivittäminen on raskas operaatio, mutta kaikkien replikoiden konsistenssi on taattu. Kevyempi ja samalla epävarmempi on

laiska replikointi (engl. *lazy replication*), jossa muutokset tehdään aluksi vain pääkopioon. Haittapuolena on se, että replikat saattavat olla hetken aikaa epäkonsistentissa tilassa, kunnes muutos tulee voimaan niissäkin. Pääkopion vikaantuessa viimeisimmät muutokset saattavat hukkua kokonaan.

2.4 Skaalautuvuus

Skaalautuvuudella (engl. *scalability*) tarkoitetaan järjestelmän kykyä kasvattaa tehokkuuttaan ennakoitavissa olevalla tavalla, kun sen käyttöön annetaan enemmän tai parempia resursseja. Rinnakkaisissa järjestelmissä termillä *vertikaalinen skaalaaminen* tarkoitetaan yksittäisen solmun tehostamista esimerkiksi laitteistopäivityksen avulla ja *horisontaalisella skaalaamisella* järjestelmän tehostamista liittämällä siihen lisää solmuja. Klusterijärjestelmien välillä vertaillaan tyypillisesti niiden horisontaalista skaalautuvuutta.

Ideaalinen arkkitehtuuri *skaalautuu lineaarisesti*, jolloin esimerkiksi klusterin tehokkuus on suoraan verrannollinen siinä olevien solmujen määrään. Järjestelmälle, jossa on N solmua, lasketaan skaalauskerroin kaavalla

$$skaalauskerroin = \frac{N\ solmun\ teho}{1\ solmun\ teho} \div N.$$

Skaalauskerroin kertoo, kuinka lineaarista skaalautuvuutta järjestelmä toteuttaa. Jos tietokantaklusteri kykenee toteuttamaan yhdellä solmulla 100 hakua sekunnissa ja kymmenellä solmulla 950 hakua, on järjestelmän kymmenen solmun skaalauskerroin 0,95. Täysin lineaarinen skaalautuvuus toteutuisi arvolla 1,0.

2.5 Kuormantasaus

Kuormantasauksella pyritään estämään tilanteita, joissa yksittäinen järjestelmän solmu ylikuormittuu samaan aikaan kuin muilla solmuilla resursseja olisi vapaana. Ilman toimivaa kuormantasausta klusterin kokonaistehoa ei saada valjastettua käyttöön, ja ongelma näkyy palvelun asiakkaille turhina viiveinä.

Perinteinen tapa toteuttaa klusterin solmujen välinen kuormantasaus on ohjata asiakkaiden palvelupyynnöt vaihtelevasti eri solmuille. Yksinkertainen staattinen kuormantasaus toteutuu, kun sovitaan ennalta, mikä palvelin palvelee mitäkin kyselyä; esimerkiksi kahdesta solmusta toinen palvelee parittomista IP-osoitteista tulevat pyynnöt ja toinen parillisista tulevat. Tällöin ei tarvita erillistä kuormantasainta, mutta kuormituksen tasaisuudesta solmujen välillä ei myöskään ole mitään takeita.

Toinen yleinen ratkaisu on hyödyntää verkon nimipalvelua asettamalla yksi konenimi vastaamaan useita IP-osoitteita, jotka on jaettu klusterin solmujen kesken. Nimipalvelu jakaa asiakkaille eri solmujen osoitteita tasaisesti jakavalla algoritmilla, kuten kiertovuorottelulla (engl. *Round Robin*) [11]. Tällaisessa ratkaisussa yhden solmun kaatuminen ei kuitenkaan poista sen osoitetta jakelusta ja osa asiakkaiden yhteydenotoista ohjautuu kaatuneen solmun osoitteeseen, kunnes vika korjataan tai nimipalvelun tietoja päivitetään.

Edellä mainittuja kehittyneemmät kuormantasaimet keräävät tietoa klusterin tilasta. Asiakkaan yhteyden vastaanottava solmu voidaan valita ottaen huomioon solmujen kulloinenkin kuormitusaste tai keskimäärin niiden viime aikoina vastaamiin palvelupyyntöihin kulunut aika. Tällaiset ominaisuudet vaativat tilallista kuormantasausta, jollaisia on saatavilla ohjelmistopohjaisten ratkaisujen lisäksi joidenkin verkkokytkimien sisäänrakennettuna ominaisuutena.

2.6 Tietokantapalvelujen suorituskyvyn mittaaminen

Tietokonejärjestelmien suorituskyvyn mittaamiseen käytetään usein erityisiä testisovelluksia (engl. *benchmark*). Ne kuormittavat järjestelmää keinotekoisella kuormalla. Samalla valvotaan kuormituksen vaikutuksia järjestelmän toimintaan. Testin tarkoituksesta riippuen voidaan olla kiinnostuneita esimerkiksi järjestelmän resurssien kulutuksesta ja tehtyjen töiden suoritusajoista. Niiden perusteella lasketaan tyypillisesti joukko numeroita, jotka kuvaavat testin tehokkuutta kyseisessä ympäristössä. Suuren kuormituksen avulla voidaan myös löytää ohjelmistovikoja, jotka matalalla kuormalla esiintyisivät harvoin tai eivät lainkaan.

Yksittäisellä koneella saatu suorituskykytestin tulos ei yleensä yksinään sisällä juurikaan hyödyllistä informaatiota. Tarvitaan vertailtavaksi toinen tulos, joka on saatu eri parametreilla, kuten eri koneella, ohjelmistolla tai testiohjelman asetuksilla. Eri koneilla saatujen tulosten erojen syyt eivät ole aina selvät: pullonkaulan voivat muo-

dostaa useat eri komponentit tai niiden yhteisvaikutus. Tämän vuoksi tuloksia on helpointa analysoida, jos testiajojen parametrit poikkeavat toisistaan vain yhdessä asiassa kerrallaan.

Tunnettu tietokantajärjestelmän suorituskykyä mittaava testi on TPC-C [29]. Se on yksi useista Transaction Processing Performance Councilin kehittämistä testeistä, jotka alunperin pyrkivät mallintamaan erästä pankkisovellusta. Tällaisia tosielämän sovellusta jäljitteleviä testejä kutsutaan nimellä *application benchmark*.

TPC-C kuormittaa testattavaa tietokantaa laajalla valikoimalla erilaisia transaktioita. Tietokantamalli sisältää paljon erilaisia tauluja ja niissä useita erikokoisia tietotyyppejä. Testi mallintaa kattavasti erilaisia toimintoja, joiden tehokkuus voisi olla tärkeää todelliselle tietokantasovellukselle.

TPC-C:n kaltaisen suositun ja pitkään käytössä olleen testin etuna on se, että valmiita vertailutuloksia on olemassa runsaasti. High-end -järjestelmien valmistajien omat TPC-C-tulokset ovat usein näkyvästi esillä tuotteiden markkinointimateriaaleissa. Ne eivät kuitenkaan kerro paljoakaan siitä, kuinka tehokas kukin järjestelmä on jonkin yksittäisen tietokantasovelluksen alustana. Tietokantoja voidaan optimoida eri asioille ja mitä enemmän käytetty sovellus poikkeaa TPC-C -testin rakenteesta, sitä vähemmän arvoa testituloksella on [4]. Arvokkaampia tuloksia saadaan käyttämällä testiä, joka muistuttaa lähemmin todellista sovellusta.

Luku 3

Toteutustekniikoita ja -välineitä

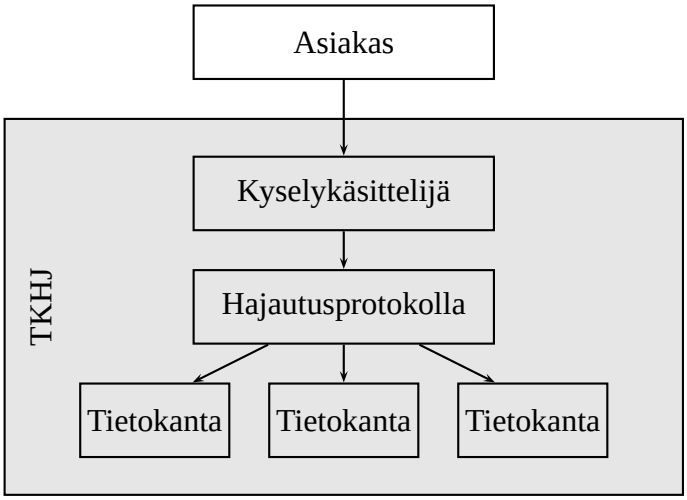
3.1 Hajautustekniikat

Yksityisen muistin arkkitehtuuriin perustuva klusterointi voidaan toteuttaa useilla eri sovelluskerroksilla. Avoimen lähdekoodin toteutuksia on olemassa erilaisista arkkitehtuureista:

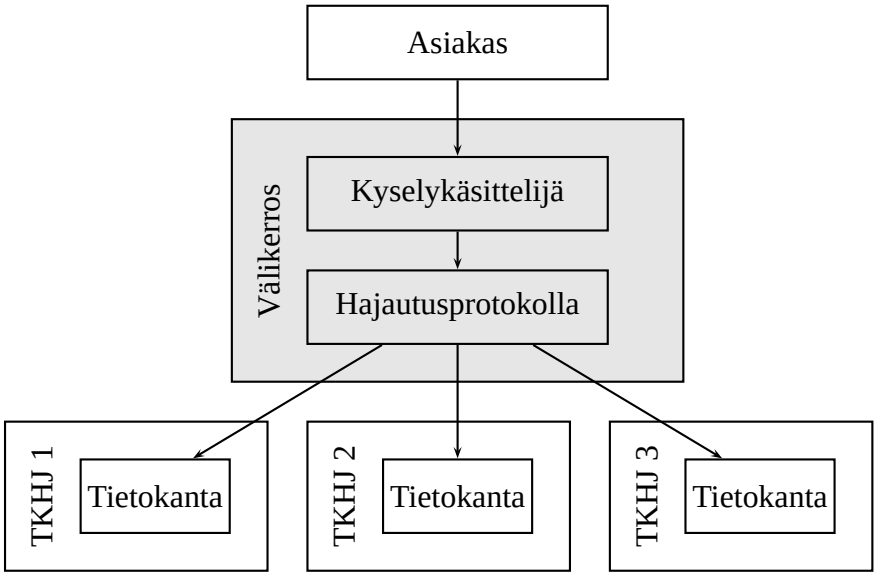
Hajautus tiedostojärjestelmässä. Tästä on esimerkkinä klusterointitiedostojärjestelmä Lustre [12]. Se toteuttaa verkon yli käytettävän tiedostojärjestelmän, jonka alustana toimii joukko palvelimia. Tiedostojen attribuutit sijoitetaan ns. *metadata*-palvelimille, mutta niiden varsinainen sisältö on oliotietokannassa, joille on omat erilliset *object storage* -palvelimensa. Vaikka yksittäinen tiedosto voi sijaita useissa osissa eri palvelimilla, sovelluksille se näkyy tiedostojärjestelmässä yhtenäisenä kappaleena.

Hajautus tietokantajärjestelmässä (kuva 3.1). Monet tietokannanhallintajärjestelmät sisältävät itsessään hajautustoimintoja. Tästä eräs esimerkki on tässä luvussa esiteltävä MySQL Cluster [17].

Hajauttava välikerros (kuva 3.2). Hajautuksen voi toteuttaa erillinen kolmannen osapuolen tekemä sovellus, joka käsittelee asiakkaiden ja palveluprosessien välistä liikennettä. Välikerrosarkkitehtuuria toteuttavat mm. Slony-I [23], joka tuo PostgreSQL -tietokannanhallintajärjestelmään [21] replikointiominaisuuksia, ja C-JDBC [3], joka mahdollistaa tietokantaklusteroinnin JDBC-ohjelmointirajapinnan kautta tietokantaa käyttäville Java-sovelluksille.



Kuva 3.1: Tietokannan hajautus tietokantajärjestelmässä.



Kuva 3.2: Tietokannan hajautus välikerroksessa.

Seuraavassa esitellään joukko ilmaisia tai edullisia ohjelmistoja, joita voidaan käyttää hajautetun tietokantapalvelun perustana. Kuvaukset perustuvat niihin ohjelmaversioihin, joita testiympäristöä rakennettaessa käytettiin. Tarkat versionumerot kerrotaan kohdassa 4.2.

3.2 MySQL

MySQL [15] on suosittu avoimen lähdekoodin relaatiotietokannanhallintajärjestelmä, jota kehittää Ruotsissa perustettu yhtiö MySQL AB. Yhtiö kertoo olevansa toisen sukupolven avoimen lähdekoodin yhtiö joka yhdistää avoimen lähdekoodin arvomaailman ja menetelmät menestyvään liiketoimintamalliin. Tällä viitataan yhtiön tarjoamaan ”dual licensing” -lisenssimalliin: MySQL on saatavilla lisensoituna sekä avoimella GPL-lisenssillä [8] että MySQL:n omalla kaupallisella lisenssillä. Kaupallinen lisenssi on maksullinen, mutta sen käyttäminen ei velvoita luovuttamaan omia MySQL:ää käyttäviä ohjelmistoja tai niiden lähdekoodeja ilmaiseen levitykseen, kuten GPL vaatii. Kaupallisen lisenssin mukana saa myös täyden tuotetuen MySQL AB:lta. Tarjoamansa maksullisen tuen ansiosta MySQL on saanut jalansijaa myös yrityskäytössä muilta avoimen lähdekoodin tietokantajärjestelmiltä.

Useimmista tietokannoista poiketen MySQL tukee useampaa kuin yhtä *tietokantamoottoria* (engl. *storage engine, table handler*). Kukin moottori näkyy asiakasohjelmalle omana taulutyyppinään. Samaan tietokantaan voidaan luoda useita erityyppisiä tauluja. Tietokantamoottorien vastuulla on tiedon säilytys ja käsittely, kun taas itse MySQL toimii asiakasohjelmien yhteyksiä välittävänä rajapintana. Tietokantajärjestelmien tehokkuutta tai käyttökelpoisuutta vertailtaessa voidaan harvoin puhua pelkän MySQL:n tukemista ominaisuuksista, vaan on myös otettava huomioon, mitä MySQL:n tietokantamoottoria käytetään.

Yleisimmin käytetyt MySQL:n moottorit ovat MySQL:n oletusmoottori, ei-transaktiollinen MyISAM sekä transaktioita tukevat, MySQL-projektin ulkopuolella kehitetyt InnoDB ja BerkeleyDB.

Useita palvelimia voidaan yhdistää tarjoamaan yhteistä tietokantaa MySQL-rajapinnan sisältämillä replikointiominaisuuksilla, jotka noudattavat laiskaa replikointia. Tietokannan pääkopion sisältävä palvelin määritellään isäntäpalvelimeksi ja muut orjapalvelimiksi (engl. *master/slave replication*). Orjat pitävät yllä kopiota isännän tietokannasta. Asiakkaat voivat lähettää lukuoperaatioita sekä isännälle että orjille, mutta

kaikki tietokantaa päivittävät operaatiot on lähetettävä isännälle. Tämä kirjoittaa tekemänsä muutokset lokiin, jota orjat lukevat ja toistavat sen mukaisesti muutokset omaan kopioonsa. Orja saattaa siis palauttaa asiakkaalle tietoa joka ei ole ajan tasalla, jos se ei ole vielä tehnyt kaikkia muutoksia omaan kopioonsa. Usein MySQL:n replikointia hyödynnetään siten, että isäntäpalvelimen kuormituksen vähentämiseksi sille ohjataan vain päivitysoperaatiot, kun taas orjapalvelimet käsittelevät kaikki lukuoperaatiot. Minkäänlaista automaattista kuormantasausta ei ole, vaan operaatioiden ohjaus oikeille palvelimille on toteutettava asiakasohjelmassa.

Isäntäpalvelimen kaatuessa orjapalvelimien toiminta ei häiriinny, mutta tietokannan päivittäminen ei enää onnistu. Lisäksi tietoja saatetaan menettää, jos uusimmat muutokset eivät vielä ole levinneet isännältä orjille. Jos isäntää ei saada heti takaisin käyttöön, voidaan yksi orjista ylentää uudeksi isännäksi, mutta se ei tapahdu automaattisesti, vaan edellyttää joko käsityötä järjestelmän ylläpitäjältä tai itse tehdyn tilanvalvontaohjelman. Myöhempiin MySQL:n versioihin on kehitteillä automaattinen isännänylennystoiminto.

Vikasietoisuutensa lisäksi replikointijärjestelmä on rajoittunut myös skaalautuvuudeltaan, koska isäntäpalvelimia voi olla vain yksi. Se ei muodostu ongelmaksi, jos kirjoitusoperaatiot muodostavat niin pienen osan järjestelmän kuormituksesta, että niille riittää yksi palvelin. Orjapalvelimia voidaan lisätä lähes rajatta.

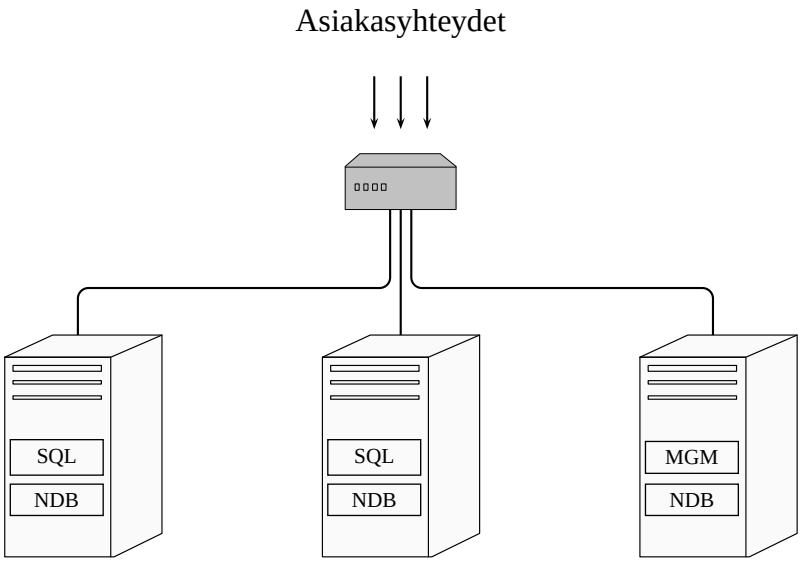
3.3 MySQL Cluster

MySQL Cluster [17] on MySQL:n mukana versiosta 4.1.3 lähtien tullut lisäosa, joka mahdollistaa yksityisen muistin arkkitehtuuriin perustuvan tietokantaklusterin rakentamisen Linux-, Mac OS X- tai Solaris-palvelimista. Tulevissa (4.1-sarjaa uudemmissa) versioissa käyttöjärjestelmätuki aiotaan laajentaa kattamaan muutkin MySQL:n tukemat alustat. Siinä missä MySQL:n sisäänrakennettu replikointi on laiska, toteuttaa MySQL Cluster ahkeraa replikointia, jossa solmut eivät voi jäädä päivitysten voimaantulossa toisistaan jälkeen. Erityistä isäntäpalvelinta ei ole, vaan klusteri jatkaa toimintaansa huolimatta minkä tahansa yksittäisen solmun kaatumisesta.

MySQL Cluster integroituu MySQL:ään hyödyntäen sen tukea vaihtoehtoisille tietokantamoottoreille. Sillä on oma transaktioita tukeva moottorinsa, Network Database (NDB). NDB:tä käytetään MySQL-rajapinnan lävitse samaan tapaan kuin InnoDB:tä

ja BDB:tä. NDB:tä suoritetaan kuitenkin MySQL:stä erillisenä prosessina. Klusterin toimintaan vaaditaan vähintään yksi instanssi kustakin seuraavista prosesseista:

- Dataprosessi (NDB) tallentaa ja replikoi tietokannassa olevaa tietoa muodostaen varsinaisen tietokantaklusterin yhdessä muiden dataprosessien kanssa.
- SQL-prosessi (MySQL) toimii asiakkaiden ja dataprosessien välillä asiakasyhteyksiä välittävänä rajapintana.
- Hallintaprosessi (MGM) hallitsee ja valvoo muita prosesseja, niiden asetuksia, käynnistämistä, pysäyttämistä, varmuuskopiointia jne.



Kuva 3.3: MySQL Cluster -verkkokonfiguraatio.

Kuva 3.3 esittää yhtä mahdollista MySQL Cluster -verkkokonfiguraatiota, mutta prosessit on mahdollista sijoitella solmujen kesken monin eri tavoin. SQL- ja dataprosesseja on tyypillisesti käynnissä useita ja ne kaikki voivat sijaita eri solmuissa, mikä täyden rinnakkaisuudesta saatavan hyödyn saavuttamiseksi onkin suositeltavaa. Myös hallintaprosessi voidaan sijoittaa täysin omalle solmulleen. Solmut liikennöivät keskenään TCP/IP-verkon välityksellä, jonka vähimmäisnopeutena on 100 Mbit/s Ethernet tai vastaava.

Datasolmut säilyttävät tietokannassa olevia tietoja. Järjestelmä on tyypiltään ns. *in-memory*, eli tieto säilytetään kokonaisuudessaan koneiden keskusmuistissa. Sen vuoksi suuren tietokannan tallennus klusteriin lisää koneiden muistintarvetta huomattavasti. Koska tieto on jaettu tasan kaikkien solmujen kesken, solmuissa käytettävän muistin määrä määräytyy pienimuistisimman koneen mukaan. Oletusasetuksilla tietokannan sisältö on jaettu solmujen yhteiseen muistiavaruuteen täysin ilman replikointia. Yksittäinen tieto on siis vain yhdessä paikassa, ja jokainen tietokantataulu on paloiteltu useaksi *pirstaleeksi* (engl. *fragment*) monelle eri solmulle. Vikasietoisuuden parantamiseksi tietokannan replikoiden määrää voidaan kuitenkin lisätä 2-4 kappaleeseen. Esimerkiksi kaksinkertainen replikointi kuluttaa luonnollisesti ainakin kaksinkertaisen määrän muistia. MySQL Cluster FAQ:ssa [16] annetaan kaava, jolla voi hyvin karkeasti arvioida datasolmun muistintarvetta:

$$solmun\ muistintarve = 1,1 \times \frac{tietokannan\ koko \times replikoiden\ määrä}{solmujen\ määrä}.$$

Datasolmuja asennettaessa määritellään kiinteä muistimäärä, jonka ne voivat käyttää tietojen tallentamiseen. Tästä alueesta varattua muistia ei vapauteta muulloin kuin poistettaessa tietokannasta kokonainen taulu. Muistialueen voi eheyttää käynnistämällä solmun uudelleen.

Jos replikointi määritellään käyttöön, klusteri jakaa automaattisesti solmut ryhmiin, joiden kooksi tulee yhtä monta solmua kuin mikä on replikoiden lukumäärä. Samaan replikointiryhmään kuuluvat solmut säilyttävät keskenään kopiota samasta tietosisällöstä. Tietokanta on kokonaan saavutettavissa, jos kustakin ryhmästä on toiminnassa vähintään yksi solmu. Tarvittavien koneiden määrää voidaan vähentää ajamalla useampaa dataprosessia samassa solmussa, mutta ko. prosessien on luonnollisesti kuuluttava eri replikointiryhmiin, jotta solmun kaatuminen ei vie mukanaan kokonaista ryhmää. Taulukossa 3.1 on kuvattu kuuden dataprosessin sijoittelu kolmelle koneelle, kun tietokanta on replikoituna kahdeksi kopioksi. Tällöin yksittäisen koneen kaatuessa jää jäljelle vielä vähintään yksi kuhunkin ryhmään A, B tai C kuuluva prosessi, mikä riittää tietokannan toimintaan.

Asiakkaan ottaessa yhteyttä SQL-solmuihin ovat voimassa vastaavat ongelmat kuin MySQL:n sisäänrakennettua replikointia käytettäessä: SQL-solmut näkyvät verkossa omina osoitteinaan, ja asiakkaan on valittava niiden joukosta kohteensa. Klusteri ei sisällä mekanismeja yhdeksi koneeksi naamioitumiseen tai SQL-solmujen väliseen kuormantasaukseen. Puutetta voidaan korjata perinteisin keinoin, kuten käyttämällä ulkoista kuormantasainta tai nimipalvelimen kiertovuorottelua. Kuormantasauksen

Taulukko 3.1: Dataprosessien sijoittelu kolmelle solmulle, kun replikoiden määrä on kaksi.

	Solmu 1	Solmu 2	Solmu 3
Ryhmä A	•	•	
Ryhmä B		•	•
Ryhmä C	•		•

tarve on kuitenkin pienempi, koska SQL-solmut eivät suorita itse tietokantamoottoria ja kuluttavat sen vuoksi vähemmän resursseja kuin itsenäinen MySQL-palvelin.

MySQL Clusterin arkkitehtuurissa tarvitaan kuormantasausta myös toisessa kyselyn vaiheessa: SQL-solmun tehdessä valintaa datasolmujen kesken. SQL- ja datasolmujen välisessä liikennöinnissä toteutuu yksinkertainen kuormantasaus automaattisesti, koska tietokantataulujen pirstaleet sijaitsevat eri solmuilla. Tällöin eri solmuja kutsutaan todennäköisesti melko tasaisesti. Tämän lisäksi SQL-prosessi voi kierrättää valintaa siitä, mihin replikaan yhteyksiä kulloinkin otetaan. Kyseinen tekniikka perustuu kuitenkin vain todennäköisyyksiin eikä kyseessä ole todellinen dynaaminen kuormantasaus.

Klusteri tukee korkeintaan 63 yhtäaikaista prosessia, joista 48 voi olla dataprosesseja. Järjestelmän skaalaus on työläs operaatio; SQL- tai hallintasolmun lisääminen vaatii koko klusterin uudelleenkäynnistämisen. Datasolmun lisääminen on jo hyvin aikaavievä prosessi: koko tietokannan sisältö on varmuuskopioitava, klusteri nollattava ja käynnistettävä uuden solmun kanssa uudelleen. Lopuksi tietokanta on populoitava uudelleen varmuuskopiosta.

3.4 Emic M/Cluster

Emic Networksin valmistama Emic M/Cluster [13] on välikerrosohjelmisto, jonka avulla joukosta MySQL/Linux-palvelimia voidaan rakentaa skaalautuva korkean saatavuuden klusteri[1]. M/Cluster on lähdekoodiltaan suljettu kaupallinen ohjelmistotuote, mutta sen luvataan säästävän monissa kuluissa verrattuna kokonaan kaupalliseen

[1]Aiemmin nimi oli Emic Application Cluster for MySQL (EAC). Nimimuutoksia on tapahtunut myös testaustyön päättymisen jälkeen. Nykyisin ohjelmistovalmistaja on nimeltään Continuent ja tietokantaratkaisu Continuent M/Cluster. Tämän vuoksi viite on eri nimiseen tuotteeseen kuin mitä tässä käytetään.

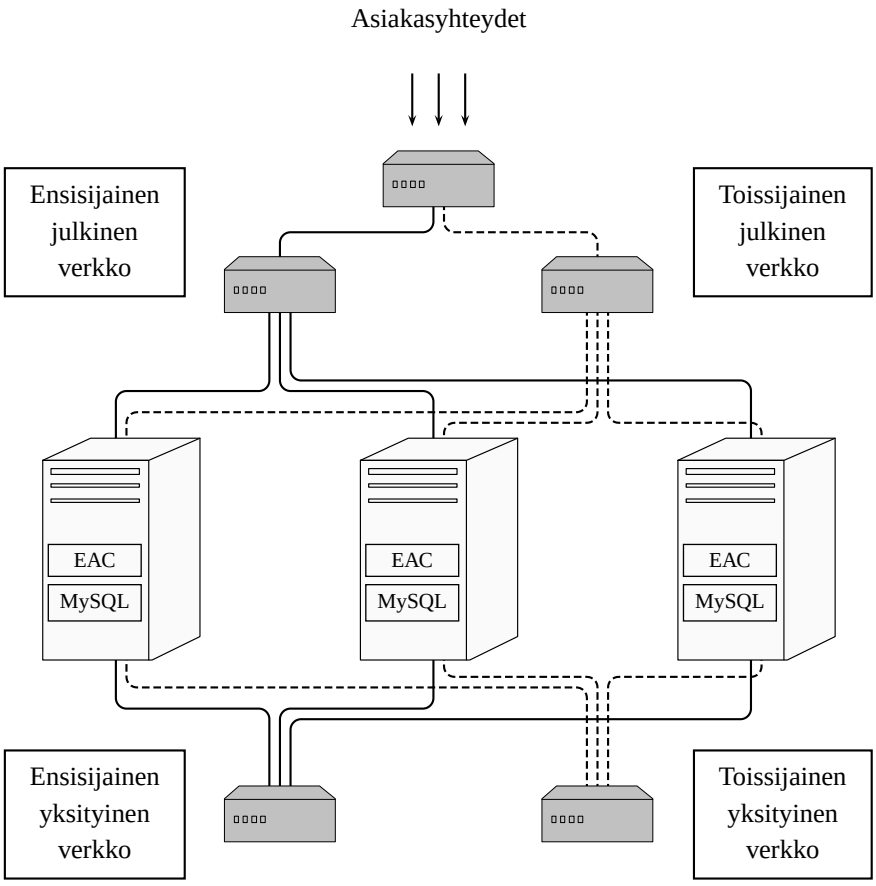
klusteriratkaisuun, kun voidaan hyödyntää yleiskäyttöisiä laitteistokomponentteja ja avoimen lähdekoodin käyttöjärjestelmää ja tietokantajärjestelmää.

Toisin kuin MySQL Cluster, M/Cluster ei muodosta klusteria MySQL-rajapinnan taakse, vaan sen eteen. Jokaisella solmulla suoritetaan kahta prosessia:

- MySQL-prosessi ei tiedä muiden solmujen olemassaolosta, vaan on kuin mikä tahansa yksittäinen MySQL-palvelin. Prosessi saa kutsuja vain paikalliselta EAC-prosessilta ja toimii vain oman solmunsa tietojen tallennuspaikkana. Tietokantamoottorina voidaan käyttää mm. MyISAM:ia ja InnoDB:tä. Prosessi on käytännössä M/Clusterin mukana toimitettava Emicin muokkaama versio MySQL-palvelinohjelmistosta, jonka lukitusten hallintaa ja kyselyjen sarjallistamismekanismeja on optimoitu.
- EAC-prosessi ottaa vastaan asiakkaiden yhteyksiä ja toteuttaa varsinaiset klusterointitoiminnot kuten replikoinnin ja kuormantasauksen yhdessä muiden solmujen EAC-prosessien kanssa. Lisäksi prosessi ohjaa tarvittavat tietokantaoperaatiot oman solmunsa MySQL-prosessille.

Esimerkki kolmen solmun M/Cluster-verkkokonfiguraatiosta on kuvassa 3.4. Asiakkaiden yhteydet tulevat klusteriin julkisen verkon kautta. Solmujen keskinäistä kommunikaatiota varten on lisäksi oma yksityinen verkkonsa. Missään ei tarvitse luottaa yksittäiseen komponenttiin, vaan molemmat verkot voidaan haluttaessa kahdentaa. Tästä syystä kuhunkin solmuun suositellaan neljää Ethernet-verkkokorttia, mutta vähimmäisvaatimus solmua kohden on kaksi.

EAC-prosessit kommunikoivat keskenään UDP-pohjaisen multicast-protokollan avulla [6]. Protokolla varmistaa, että viestit menevät perille kaikkiin solmuihin. Asennusvaiheessa voidaan valita, halutaanko käyttöön tiukka vai heikko kyselyjen konsistenssikäytäntö (engl. *strict/weak query consistency policy*). Tiukka käytäntö tarkoittaa, että tietokantapäivitykset suoritetaan solmuissa keskinäisesti samassa järjestyksessä ja ahkeraa replikointia toteuttavalla algoritmilla varmistetaan, että solmujen konsistenssi säilyy jokaisen päivityksen jälkeen. Päivityksestä lähetetään asiakkaalle kuittaus vasta, kun tieto on mennyt perille kaikkiin aktiivisiin replikoihin. Heikko käytäntö muistuttaa MySQL:n sisäänrakennettua replikointiprotokollaa sikäli, että klusterin solmut voivat ajoittain palauttaa eri versioita tietokannan sisällöstä. Heikko käytäntö skaalautuu paremmin lukuoperaatiopainotteisessa käytössä, mutta se on käyttökelpoinen vain sellaisissa sovelluksissa, joissa ei haittaa, että haut voivat palauttaa vanhentunutta tietoa.



Kuva 3.4: M/Cluster-verkkokonfiguraatio.

Klusterin solmujen maksimimäärää ei ole ilmoitettu, mutta kullekin annetaan yksilöllinen tunnusluku väliltä 0-15, mikä rajoittaa klusterin koon korkeintaan 16 solmuun. Kukin solmu pitää yllä omaa kokonaista replikaa tietokannan sisällöstä, joten yksittäisen solmun kaatuminen ei kaada koko klusteria. Uusien solmujen lisääminen klusteriin ei vaadi palvelun katkaisemista, vaan se voidaan tehdä ”lennossa”, jos klusterissa on ennestään vähintään kaksi toimivaa solmua. Kahden solmun vaatimus johtuu siitä, että uuden solmun liittyessä yksi vanhoista solmuista joutuu keskeyttämään normaalin toimintansa ja siirtymään synkronointitilaan, jossa se lukitsee tietokantansa senhetkisen tilan ja kopioi sen liittyjälle. Kopioinnin valmistuttua solmut replikoivat siirron aikana tulleet uudet päivitykset tietokantaan ja siirtyvät lopuksi aktiiviseen tilaan. Synkronointiprotokollassa on myös tila, jossa synkronoinnin aikana lukitaan vain yksi taulu kerrallaan, mutta se on käytettävissä vain, jos ei käytetä useita tauluja käsitteleviä transaktioita.

Klusteri naamioituu yksittäiseksi koneeksi siten, että kaikki solmut käyttävät julkisessa verkossa yhteistä virtuaalista IP-osoitetta, jonka kanssa asiakkaat kommunikoivat. Tällöin asiakkaan ei tarvitse tietää mitään klusterin rakenteesta eikä solmujen lukumäärän muutoksista. Solmut sopivat automaattisesti joukostaan yhden delegoijasolmun, joka ottaa asiakkaiden pyynnöt vastaan ja delegoi ne kaikkien solmujen kesken kuormantasaussääntöjen mukaisesti.

Kuormantasaussääntöjen avulla klusteri pyrkii tasapainottamaan solmujen keskinäistä kuormitusta. Yhteyden kohdesolmu määritetään funktiolla, jonka parametreina käytetään asiakkaan ja palvelimen IP-osoitteita sekä yhteydessä käytetyn TCP/UDP-portin numeroa. Tämän lisäksi kuormantasaimessa on dynaamisia ominaisuuksia: kunkin solmun kuormitusta valvotaan säännöllisillä mittauksilla. Jos jokin solmu joutuu kovaan kuormitukseen, delegoija ohjaa yhteyksiä enemmän muille solmuille, joilla on vapaita resursseja. Kuormitusmittaukset koskevat vain suorittimen käyttöastetta, mutta myöhempiin M/Clusterin versioihin on luvattu monipuolisempia mittaustapoja, jotka ovat käyttäjän muokattavissa.

Toisin kuin MySQL Cluster ja osa MySQL:n tietokantamoottoreista, M/Cluster ei tue usean peräkkäisen operaation liittämistä yhdeksi transaktioksi, vaan tietokantakutsujen käsittely tapahtuu aina ns. autocommit-tilassa, jossa jokainen onnistunut SQL-lause tekee päivityksen tietokantaan välittömästi eikä sitä voida perua ROLLBACK-lauseella.

3.5 Linux-jakelut

Linux-pohjaisissa hajautusjärjestelmissä korkeaa saatavuutta tavoiteltaessa sopivan käyttöjärjestelmäjakelun valinnalla voi olla merkitystä. Tärkeintä on tietysti se, että tarvittavat palvelinohjelmistot ylipäätään saadaan toimimaan. Toimivan järjestelmän rakentamisen jälkeen olennaisia kriteereitä voivat olla mm. jakelun päivitysmekanismit sekä jakelun valmistajan tarjoama tukipalvelu. Tietoturvapäivitykset eivät välttämättä ole kriittisiä, sillä klusterikoneita ei kytketä julkiseen verkkoon. Ne pidetään turvatussa sisäverkossa, jossa koneiden tehokkuutta ja toimintaa ei haluta riskeerata turhilla päivityksillä. Erilaisia Linux-jakeluita on useita, mutta tässä esitellään vain ne jotka olivat tämän työn kannalta oleellisia.

Red Hat Enterprise Linux (RHEL) [22] on Red Hat Inc:n yrityskäyttöön suunnattu Linux-jakelu. RHEL:ää saa palvelinkäyttöön kolmella erilaisella tukipalvelutasolla,

joista kaikki ovat maksullisia. Red Hat on yrityskäytössä selkeä markkinajohtaja, mikä johtuu osaksi siitä että on saanut hallita pitkään markkina-aluettaan lähes yksin.

Fedora Core [7] on Red Hat Inc:n rahoittaman Fedora Foundationin ohjauksessa kehitettävä ilmainen Linux-jakelu. Kuten ilmaisjakeluissa yleensä, osan kehitystyöstä tekee joukko vapaaehtoisia ”avoimen lähdekoodin yhteisöön” kuuluvia ihmisiä. Fedora Core toimii Red Hatille eräänlaisena testialustana kaupallisia tuotteitaan varten, sillä RHEL:n uudet versiot rakennetaan sen pohjalle. Fedora Coressa on lukuisia RHEL:ää varten kehitettyjä työkaluja, jotka pyrkivät helpottamaan käyttäjän vaivaa tarjoamalla graafisen käyttöliittymän ja paljon automatiikkaa.

Debian [5] on ilmainen Linux-jakelu, jota kehitetään vapaaehtoisvoimin. Sitä ohjaava voittoa tavoittelematon organisaatio noudattaa jakelun kehityksessä avointa ja epäkaupallista ideologiaa. Debianin asennus ja ylläpito vaatii moniin yritysvetoisiin Linuxeihin verrattuna käyttäjältä enemmän ymmärrystä järjestelmän toiminnasta ja alustana toimivasta laitteistosta. Se on suosittu käyttöjärjestelmä sekä harrastaja- että yrityspalvelimissa. Debian on saatavilla kolmena eri julkaisuhaarana: *stable*, *testing* ja *unstable*. Kun testing-vaiheessa oleva Debian-versio on kehittynyt riittävän vakaaksi, se muutetaan Debian Projectin päätöksellä stableksi ja edellisen stable-version kehittäminen lopetetaan. Samalla osasta unstablea erotetaan uusi testing. Kaikilla Debian-versioilla on oma nimensä, joka pysyy samana sekä testing- että stable-vaiheessa. Tätä työtä tehtäessä testing-vaiheessa oli Debian Sarge ja stable-vaiheessa Debian Woody eli Debian GNU/Linux 3.0.

Luku 4

Testiympäristö

4.1 Sovellusalue

4.1.1 HLR-tietokanta

Eräs tietokantasovellus, jonka käyttöaste kasvaa jatkuvasti, on mobiilikommunikaatioverkon käyttämä Home Location Register (HLR). Siihen tallennetaan mobiiliyhteyksien kytkemiseen tarvittavia tietoja: puhelinnumeroita, asiakastietoja sekä erilaisten lisäpalvelujen asetuksia, kuten soitonsiirtojen kohdenumeroita. Tietokannasta haetaan tilaajatietoja aina puhelua yhdistettäessä. Palvelujen käyttäjät ovat tottuneet perinteisen puhelinverkon toiminnan sujuvuuteen, eikä useiden sekuntien mittaisia viiveitä suvaita langattomasti kommunikoitaessakaan.

HLR:n erityispiirteet moniin muihin tietokantasovelluksiin verrattuna ovat seuraavat:

- Tietokanta on rakenteeltaan suhteellisen yksinkertainen.
- Tietokanta sisältää paljon rivejä, ts. tilaajia on paljon, ja määrä kasvaa nopeasti.
- Tietokantaan kohdistuu enimmäkseen lukuoperaatioita. Päivitykset ovat harvinaisempia.
- Tietokantahaut ovat kriittisiä matkapuhelinverkon sujuvalle toiminnalle, eivätkä ne saa kestää pitkään.

- Varmistukset ja varajärjestelmät ovat tärkeitä, sillä käyttökatkoja tai huoltotaukoja ei saisi esiintyä lainkaan.

Skaalautuvuus- ja saatavuusvaatimusten vuoksi esimerkiksi kaupallisten mobiilioperaattorien HLR-järjestelmät rakennetaan käytännössä hajautetun tietokantaratkaisun avulla. Saatavuuden maksimoimiseksi tietokantapalvelu olisi hajautettava klusteroinnin lisäksi myös maantieteellisesti useisiin toimipisteisiin, mutta sen simulointi sivuutettiin tässä testiympäristössä.

4.1.2 TM1 Benchmark

Telecom One (TM1) Benchmark [27] on ensimmäinen julkinen tietokantatesti, jolla mitataan tietokannan suorituskykyä telekommunikaatiokäytössä. Testi on kuvattu pro gradu -tutkielmassa [26], jonka mukaan se perustuu Nokia Networksissa sisäisesti kehitettyyn tietokantatestiin. Tiedonhallintatuotteita valmistava Solid [24] on asettanut testin toteuttavan sovelluksen lähdekoodeineen julkisesti saataville[1]. Testistä on saatavissa valmiiksi käännetty versio Windowsiin ja Linuxiin. TM1 mallintaa tyypillistä mobiiliverkon HLR-rekisterin toimintaa. Sen avulla voidaan vertailla erilaisten relaatiotietokantajärjestelmien soveltuvuutta telekommunikaatiojärjestelmiin ja arvioida laitteistokapasiteetin tarvetta. Julkisen testin toivotaan vievän teleoperaattoreiden järjestelmien kehitystä eteenpäin, sillä aiemmin operaattorit ovat yleensä tehneet testinsä itse eikä vertailukelpoisia tuloksia ole ollut saatavilla. Solid on julkistanut omasta tietokantajärjestelmästään mitattuja tuloksia ja kannustaa muita valmistajia tekemään samoin.

TM1:n ajamiseen tarvitaan seuraavat komponentit:

- Kohdetietokanta: Testattava relaatiotietokantajärjestelmä.
- TM1-testi: Tietokannan populointi, testin kontrolli ja asiakasprosessien ohjaus.
- Tulostietokanta: Test Input and Result Database (TIRDB), johon testin tulokset ja muita tietoja testiajoista tallennetaan.

[1] Myöhemmin myös Nokia on julkistanut oman testisovelluksensa ja testin spesifikaatiodokumentin avoimen lähdekoodin projektina nimeltä Network Database Benchmark [19].

Tyypillisesti edellä määritellyt komponentit sijoitetaan kukin omalle tietokoneelleen, jotka yhdistetään toisiinsa nopealla paikallisverkolla. Testi käynnistetään ajamalla TM1-ohjelma, joka avaa tarpeen mukaan yhteyksiä kohde- ja tulostietokantoja ylläpitäville palvelimille. Molempiin tietokantoihin TM1 ottaa yhteyden ODBC-rajapintaa [20] hyödyntäen, minkä ansiosta molempiin käyttötarkoituksiin voidaan hyödyntää useimpia olemassa olevia tietokantajärjestelmiä.

Yksittäinen TM1-ajo koostuu seuraavista peräkkäisistä vaiheista:

1. Aluksi tiedot generoidaan ja kohdetietokanta populoidaan.
2. Lepovaiheessa testi odottaa jonkin aikaa, jotta tietokannan tilan tasaantuu populoinnin aiheuttamasta kuormituspiikistä.
3. Kiihdytysvaiheessa testikuormitus aloitetaan ja tietokannan ja testiä ajavan koneen tilojen annetaan tasaantua kuormituksen alla. Tuloksia ei vielä mitata.
4. Testivaiheen aikana kuormitusta jatketaan tuloksia mitaten ja niitä tulostietokantaan tallentaen.
5. Lopuksi testikuormitus pysäytetään ja viimeiset tulokset koostetaan tulostietokantaan.

Kohdetietokanta sisältää neljä taulua, joiden sisältö generoituu automaattisesti. Kannan koko määräytyy HLR-rekisterin tilaajien lukumäärän mukaisesti. Kun tilaajataulussa on N tilaajan populaatio, muissa tauluissa on yhteensä $8,75 \times N$ riviä. Taulujen rivit täytetään ennalta määrättyjen jakaumien mukaisesti ja tilaajien tietoihin tuotetaan satunnaisia lukuja ja merkkijonoja. Rakenne kuvataan tarkemmin lähteessä [28].

Itse testaus tapahtuu siten, että testikoneella käynnistyy haluttu määrä asiakasprosesseja, jotka lähettävät transaktioita kohdetietokannalle. Kukin asiakasprosessi valitsee lähetettävän transaktion määrättyjen todennäköisyyksien mukaisesti, jolloin niiden lukumäärät noudattavat haluttua jakaumaa. Transaktiot ovat ennalta määriteltyjä, mutta asiakasprosessit täyttävät osan niiden sisältämien operaatioiden parametreista satunnaisarvoilla. Sen vuoksi asiakasprosessit voivat yrittää toisinaan hakea tietoja avaimella, jota ei ole tietokannassa. Tällöin tietokanta palauttaa virheilmoituksen, mutta testi ei ota sitä huomioon, vaan käyttäytyy samalla tavoin kuin transaktion onnistuessa. Testi keskeytetään vain vakavammissa virhetilanteissa, esimerkiksi jonkin asiakasprosessin menettäessä yhteyden tietokantaan.

Vaikka kaikki asiakasprosessit ovat suorituksessa samalla tietokoneella, kohdetietokanta käyttäytyy kuin yhteydet tulisivat eri koneilta ja antaa kullekin asiakasprosessille vastauksia erikseen. Asiakasprosessit mittaavat jokaisessa transaktiossa vastauksen saamiseen kuluneen ajan ja tallentavat sen transaktion mukaan lajiteltuna tulostietokantaan millisekunnin tarkkuudella. Lisäksi tallennetaan kaikkien asiakasprosessien yhteensä kutakin sekuntia kohti tekemien onnistuneiden transaktioiden lukumäärä (*MQTh, Maximum Qualified Throughput*).

TM1 oli juuri julkaistu, kun sen versio 1.0 valittiin tässä työssä käytettäväksi. Testeillä ei pyritty luomaan TM1:n yhteydessä julkaistujen tulosten kanssa vertailukelpoisia lukuja, vaan päätettiin ajaa oma valikoima erilaisia testejä muutamalla eri tietokanta-alustalla ja tuottaa keskenään vertailukelpoisia tuloksia. Koska testit eivät olleet sidottu ulkopuolisiin tuloksiin, voitiin TM1:tä tarvittaessa muokata omien tarpeiden mukaan.

4.2 Tietokanta-alustat

Tietokantapalvelimina toimivat kolme samanlaista kolmen gigahertsin Pentium4-prosessorilla ja kahden gigatavun muistilla varustettua PC-työasemaa. Kussakin oli tietokannan tallennustilana erillinen 7200 rpm SATA-väylään liitetty yhdeksän millisekunnin keskimääräiseen hakuaikaan kykenevä kiintolevy. Sekä tietokantalevyllä että käyttöjärjestelmälevyllä oli tiedostojärjestelmänä Ext3. Swap-osiolle oli varattu käyttöjärjestelmälevyltä kaksi gigatavua tilaa.

TM1:tä ja sen tulostietokantaa ajettiin IBM xSeries 345 -palvelimella, joka oli varustettu kahdella kolmen gigahertsin Xeon-prosessorilla ja kahden gigatavun muistilla. Vaikka laite on tarkoitettu palvelinkäyttöön, tässä järjestelmässä se toimi asiakaskoneena. Tehokas kone valittiin, jottei itse testiohjelman vaatima koneteho muodostuisi testitulosten pullonkaulaksi, ja voitaisiin keskittyä testattavien tietokantojen kuormituksen tarkkailuun. TM1:n käyttämiä ODBC-yhteyksiä varten koneeseen asennettiin MySQL AB:n ilmainen ODBC-ajuri MySQL Connector/ODBC 3.51 [18].

Laitteiden verkkokortit kytkettiin toisiinsa 100 Mbit/s Fast Ethernet -kytkimien välityksellä. Verkon kuormitus ei missään tehokkuustestien vaiheessa noussut edes puoleen niiden mahdollistamasta kapasiteetista, joten nopeampia verkkoyhteyksiä ei pidetty tarpeellisina.

Ajan säästämiseksi tietokantajärjestelmien asetusten perusteellisen optimoinnin sijasta päätettiin turvautua tuotteiden asetusten oletusarvoihin. Jos valmistaja tarjosi valmiita erilaisille laitteistoille sopivia asetusprofiileja, niistä valittiin käytettyä laitteistoa lähimmin vastaava.

Testeihin valittiin kolme tietokantatuotetta, joista kukin muodosti yhden tietokanta-alustan:

- MySQL: Yksittäisellä palvelimella toimivan tietokannan mittaaminen ei sisältynyt työn aiheeseen, mutta perus-MySQL otettiin mukaan referenssijärjestelmäksi, jotta saataisiin samalla laitteistolla ajettuja vertailutuloksia. Myös testien toimivuus tarkistettiin ensimmäisenä MySQL:llä, koska sitä pidettiin valituista tietokanta-alustoista luotettavimpana. Testeissä käytettiin versiota MySQL 4.0.23-max for i686, joka vastasi lähimmin Emic M/Clusterin käyttämää versiota. Konfigurointiin käytettiin MySQL:n valmista asetustiedostoa my-huge.cnf, joka on tarkoitettu 1-2 gigatavun muistilla varustetuille koneille.

- MySQL Cluster: Jotta voitaisiin muodostaa kunnollinen kolmen solmun rinnakkaisuutta hyödyntävä klusteri, tarvittaisiin periaatteessa enemmän palvelimia kuin mitä testilaitteistossa oli käytettävissä: kolme datasolmua ja omat solmunsa SQL- ja hallintaprosessille. Sijoittamalla useita prosesseja samoille solmuille testauskelpoinen kolmisolmuinenkin järjestelmä voitiin kuitenkin pystyttää. Koska MySQL Cluster on mukana MySQL:ssä vasta 4.1-sarjassa, käytettiin uusinta versiota 4.1.10-max for i686.

- Emic M/Cluster: Tuotteen laitteistovaatimukset täyttyivät ja kolmen solmun klusteri voitiin rakentaa. Asennusvaiheessa valittiin käyttöön tiukka kyselyjen konsistenssikäytäntö. Solmujen tietokantamoottorina oli tuotteen mukana toimitettu MySQL:n versio 4.0.23-emic-0.6i-062 for i686 (Emic serialized version of MySQL server).

Rajoittuneisuutensa vuoksi MySQL:n sisäänrakennettu replikointi ei tässä työssä ollut kiinnostava vaihtoehto, eikä sitä otettu mukaan testeihin.

Kaikissa testeissä oli käytettävä samaa käyttöjärjestelmäjakelua, jotta jakelujen ja ohjelmistoversioiden erot eivät vaikuttaisi suorituskykyyn. Ensimmäinen valinta jakeluksi oli Fedora Core 3. Se vaikutti lupaavalta, koska järjestelmän asennus oli helppoa. Asennusohjelma tunnisti asiakaskoneen ja palvelimien komponentit automaattisesti siitä huolimatta että koneet olivat tuoreehkoa mallia, vaikka jo ennalta oli tiedossa että kyseisillä konemalleilla joidenkin jakeluiden asennus ei ole ollut ongelmaton.

Lisäksi positiivisena puolena pidettiin sitä, että Fedora muistuttaa todellisissa tuotantoympäristöissä yleensä käytettävää Red Hat Enterprise Linuxia. Ytimenä oli Fedora Core 3:n oletuskernel, versio 2.6.9-1.667smp.

M/Cluster saatiin käyttöön vasta jakelun valinnan jälkeen, jolloin asennusohjeesta selvisi, ettei Fedora ollut M/Clusterin tuettujen käyttöjärjestelmien listalla. Uudehkoista jakeluista listalla olivat vain Red Hat Enterprise Linux 3, SUSE Enterprise Server 9 ja Debian Sarge. Fedoraa kokeiltiin tuen puutteesta huolimatta, ja se todellakin tuotti ongelmia M/Clusterin kanssa: kuormituksen kasvaessa klusterin palvelimet menettivät yhteyden toisiinsa tai kaatuilivat satunnaisesti. Emic Networksin tuki vahvisti, että ongelma on Fedoran ja M/Clusterin yhteistoiminnassa. Tämän vuoksi palvelinkoneisiin asennettiin Fedoran sijasta Debian Sarge (ytimen versio 2.6.8-2, jakelun ohjelmistopaketit päivitetty kesäkuussa 2005). Tämä korjasi ongelman.

Todellisessa tuotantoympäristössä korkean saatavuuden klusteria varten olisi kahdennettava kaikki laitteiston osat, mukaan lukien verkkoyhteydet, mutta testiympäristössä ei näin pitkälle vietyjä varmistuksia pidetty tarpeellisina. Keinotekoiset testit ovat toistettavissa uudelleen, jos jokin menee vikaan, eikä varajärjestelmien olemassaololla yleensä ole vaikutusta suorituskykyyn muulloin kuin vikatilanteissa.

4.3 Testiparametrit

Hajautetussa järjestelmässä tuloksiin vaikuttavia muuttujia on enemmän kuin yksittäisen tietokantakoneen järjestelmissä. Erilaiset testiolosuhteet muodostettiin vaihtelemalla seuraavien parametrien arvoja:

Solmujen lukumäärä. Solmujen lukumäärän lisäämisellä pyritään kasvattamaan järjestelmän kokonaistehoa. Klusterituotteet testattiin yhden, kahden ja kolmen solmun kokoisina klustereina. Yksisolmuisenkin klusterin tehokkuudella on käytännön merkitystä mm. vikatilanteessa, jossa muut solmut eivät toimi. Yksittäisen palvelinkoneen toiminta testattiin myös MySQL:llä vertailulukujen tuottamiseksi.

Populaation koko. Tietokanta poistettiin, luotiin ja populoitiin uudelleen aina testikokoonpanoa tai populaatiokokoa vaihdettaessa. Testit tehtiin populaatiokooltaan sadan tuhannen, miljoonan ja viiden miljoonan tilaajan tietokannoille. Suurten populaatioiden kirjoittaminen tietokantaan osoittautui hyvin aikaavieväksi operaatioksi. Sen vuoksi suurin testattu tietokanta oli populoitu vain viidellä miljoonalla tilaajalla, vaikka tallennuskapasiteettia olisi riittänyt suuremmallekin tietokannalle. Tieto-

kannan koon kasvattaminen hidastaa yleensä sen toimintaa, koska hakuja tehtäessä joudutaan käymään läpi enemmän rivejä. Tietokannan tehokkuus voi muuttua huomattavasti siinä pisteessä, kun tietokanta ei enää mahdu palvelimen keskusmuistiin ja osa siitä joudutaan lukemaan kiintolevyltä. Tällainen tilanne oli voimassa testiympäristössä, kun populaatiokoko oli viisi miljoonaa.

Asiakasprosessien määrä. Asiakasprosessien määrällä pyritään vaikuttamaan testitietokannan kuormitusasteeseen. Liian alhaisella prosessimäärällä tietokanta ei välttämättä kuormitu vastaavalla teholla kuin todellisessa HLR-sovelluksessa. Voi myös syntyä tilanne, jossa tietyllä prosessimäärällä saavutetaan palvelimen täysi kapasiteetti, ja lisäprosessit kasvattavat vain palvelua odottavien prosessien jonoa lisäämättä varsinaista tietokannan kuormitusta [1]. TM1 rajoitti yhtäaikaisten asiakasprosessien maksimimäärän 100 kappaleeseen. Sitä suuremmalla arvolla testiohjelma ei saanut tuloksia tallennettua, vaan kaatui. Virhe oli luultavasti testiohjelmassa, sillä vian esiintyessä myös ohjelman ruudulle tulostamissa tekstiriveissä esiintyi kosmeettisia virheitä. Vian olemassaoloa ei kuitenkaan saatu varmistettua testin julkaisijalta. Testattaviksi parametrin arvoiksi valittiin 10, 50 ja 100 asiakasprosessia.

Luku- ja kirjoitusoperaatioiden suhteellinen määrä. Kirjoitusoperaatioiden lisääntyminen yleensä hidastaa tietokannan toimintaa, koska kirjoittaminen vaatii lukitusoperaatioita ja enemmän solmujen välistä kommunikointia. Eri replikointiprotokollat selviävät kirjoituskuormasta vaihtelevin tuloksin [10]. Testiohjelmassa suhde oli vapaasti muokattavissa. TM1:n mukana toimitettavissa esimerkkiasetuksissa annetaan transaktiovalikoima, jossa lukuoperaatioiden osuus on 80 prosenttia. Valikoimasta muokattiin lisäksi versiot, joissa lukuosuudet ovat 70 ja 90 prosenttia. Niissäkin luku- ja kirjoitusoperaatioiden keskinäiset lukumääräsuhteet ovat samat kuin alkuperäisessä valikoimassa. Näin testiparametrille saatiin kolme erilaista arvoa (Lukusuhteet 70 %, 80 % ja 90 %). Valikoimat on esitetty transaktiojakautumineen taulukossa 4.1.

Tietokanta-alusta. Testien kiinnostavin parametri oli klusteroinnin toteuttavan ohjelmiston vaikutus testituloksiin. MySQL Cluster ei toiminut testilaitteissa riittävän vakaasti. Epäselväksi jäi, johtuiko virhe käyttöjärjestelmän vai MySQL Clusterin viasta. Yksi sen dataprosesseista kaatui sisäiseen virheeseen joka ajokerralla, kun klusteria oli kuormitettu testiohjelmalla jonkin aikaa. Tämän vuoksi MySQL Cluster jouduttiin jättämään testien ulkopuolelle. Testattaviksi alustoiksi siis jäivät vain yhden palvelimen MySQL sekä yhden, kahden ja kolmen solmun M/Cluster.

Taulukko 4.1: Testeissä käytetyt transaktiovalikoimat.

Tyyppi	Transaktion nimi	Lukusuhde		
		70 %	80 %	90 %
luku	GET_SUBSCRIBER_DATA	31	35	39
	GET_NEW_DESTINATION	8	10	12
	GET_ACCESS_DATA	31	35	39
kir-joitus	UPDATE_SUBSCRIBER_DATA	3	2	1
	UPDATE_LOCATION	21	14	7
	INSERT_CALL_FORWARDING	3	2	1
	DELETE_CALL_FORWARDING	3	2	1

4.4 Testikokoonpanot ja -konfiguraatiot

Tuotteet testattiin kattavasti kaikilla testiparametrien arvojen yhdistelmillä, jotta tuloksista voitaisiin nähdä minkä tahansa parametrin vaikutus tietokannan toimintaan. Tällöin tuotteiden välillä voitaisiin etsiä sellaisia testiparametrien arvoja, että niiden ylittyessä testin tehokkuus muuttuisi ratkaisevasti. Koska erilaisia parametrien arvojen yhdistelmiä on paljon, otettiin niille käyttöön erityinen nimeämiskäytäntö.

Jatkossa termillä *testikokoonpano* tarkoitetaan yhdistelmää niiden parametrien arvoista, joiden muuttaminen testiajojen välissä vaatii tietokannan populoimisen uudelleen. Nämä parametrit ovat tietokanta-alusta, solmujen lukumäärä ja populaation koko. Kaikki kaksitoista testeissä käytettyä testikokoonpanoa on lueteltu ja nimetty taulukossa 4.2.

Termi *testikonfiguraatio* tarkoittaa yhdistelmää, johon kuuluu yksi testikokoonpano ja arvot lopuille kahdelle parametrille: asiakasprosessien lukumäärä sekä luku- ja kirjoitusoperaatioiden suhde. Kokoonpanon Standalone-100k konfiguraatiot on nimetty taulukossa 4.3. Myös yhdelletoista muulle kokoonpanolle on vastaavat yhdeksän konfiguraatiota. Tällöin testikonfiguraatioita on yhteensä 108.

4.5 Testiohjelman muutokset ja asetukset

Kaikki TM1:n tietokantaoperaatiot ovat pienempiä osaoperaatioita sisältäviä transaktioita, minkä vuoksi testi ei toimi M/Clusterin kanssa. M/Cluster palauttaa virheilmoi-

Taulukko 4.2: Testikokoonpanot.

Testikokoonpano	Tietokanta-alusta	Solmujen lkm	Populaation koko
Standalone-100k	MySQL	1	100 000
Standalone-1M	MySQL	1	1 000 000
Standalone-5M	MySQL	1	5 000 000
MCluster1-100k	Emic M/Cluster	1	100 000
MCluster1-1M	Emic M/Cluster	1	1 000 000
MCluster1-5M	Emic M/Cluster	1	5 000 000
MCluster2-100k	Emic M/Cluster	2	100 000
MCluster2-1M	Emic M/Cluster	2	1 000 000
MCluster2-5M	Emic M/Cluster	2	5 000 000
MCluster3-100k	Emic M/Cluster	3	100 000
MCluster3-1M	Emic M/Cluster	3	1 000 000
MCluster3-5M	Emic M/Cluster	3	5 000 000

Taulukko 4.3: Testikokoonpanon Standalone-100k testikonfiguraatiot.

Testikonfiguraatio	Asiakasprosessien lkm	Lukuoperaatioiden suhde
Standalone-100k-10-70%	10	70 %
Standalone-100k-10-80%	10	80 %
Standalone-100k-10-90%	10	90 %
Standalone-100k-50-70%	50	70 %
Standalone-100k-50-80%	50	80 %
Standalone-100k-50-90%	50	90 %
Standalone-100k-100-70%	100	70 %
Standalone-100k-100-80%	100	80 %
Standalone-100k-100-90%	100	90 %

tuksen jo siinä tilanteessa, kun sille yritetään syöttää transaktion aloittavaa BEGIN- tai lopettavaa COMMIT-lausetta. M/Clusterin saamiseksi mukaan testiin TM1:n lähdekoodia oli muokattava siten, että se toimi autocommit-tilassa eikä yrittänyt syöttää tietokannalle mitään sellaista, jota M/Cluster pitäisi virhetilanteena. Muutokset tehtiin poistamalla lähdekoodista sopimattomat rivit. Muokkaamisen tuloksena syntyneen testin ”Non-transactional TM1” (NTTM1) toiminta poikkeaa alkuperäisestä TM1:stä seuraavasti:

- TM1 kytkee populointivaiheen jälkeen autocommit-tilan pois päältä. NTTM1 ei tee tätä tilanvaihdosta, vaan jättää autocommitin päälle koko testin ajaksi.
- Jokaisen transaktion osaoperaatiot ajetaan tietokantaan alkuperäisessä järjestyksessä, mutta erillisinä toisistaan riippumattomina operaatioina.
- Ennen transaktiota ei ajeta BEGIN-lausetta eikä sen jälkeen COMMIT-lausetta.
- TM1 ajaa ROLLBACK-lauseen aina seuraavissa tilanteissa:
  - Testattava tietokanta palauttaa virheilmoituksen jollekin operaatiolle.
  - Päivitysoperaation tuloksena yksikään tietokannan rivi ei päivity.
  - Lukuoperaatio ei palauta yhtäkään riviä.

  NTTM1 jättää edellä kuvatuissa tilanteissa ROLLBACK:n ajamatta.

ROLLBACK:ien pois jääminen ei vaaranna tietokannan eheyttä, sillä kaikki testissä käytetyt kirjoitustransaktiot sisältävät vain yhden päivitysoperaation, ja se on aina transaktion osaoperaatioista viimeinen. Ainoana poikkeuksena on kahdesta päivitysoperaatiosta koostuva transaktio UPDATE_SUBSCRIBER_DATA (Kuva 4.1). Tilaajataulun sarake `bit_1` saa satunnaisen arvon jo tietokantaa populoitaessa, eikä sen arvoa hyödynnetä muissa päivitystransaktioissa. On siis helposti nähtävissä, että vaikka transaktio keskeytettäisiin ensimmäisen operaation jälkeen, tietokannan eheys säilyy.

Testien välistä vertailua varten jokaiselle ajolle laskettiin *AQTh* (*Average Qualified Throughput*, keskimääräinen MQTh sekuntia kohti). Näin saatiin yksittäinen lukuarvo, jonka perusteella ajojen eroja ja niiden onnistumista oli helpompi arvioida.

NTTM1:tä ja alkuperäistä TM1:tä vertailtaessa huomattiin, että NTTM1:llä saavutettiin parhaimmillaan yli 20 prosenttia korkeampi AQTh kuin TM1:llä. Paremmuus korostui sitä enemmän, mitä lukupainotteisempi transaktiovalikoima oli. Tulosten eron

```
UPDATE  subscriber
SET     bit_1 = <satunnainen bitin arvo>
WHERE   s_id = <satunnainen tilaaja-id>

UPDATE  special_facility
SET     data_a = <satunnainen arvo>
WHERE   s_id = <edell. lauseen tilaaja-id> AND
        sf_type = <satunnainen arvo>
```

Kuva 4.1: Transaktio UPDATE_SUBSCRIBER_DATA.

pääteltiin johtuvan NTTM1:n käyttämästä autocommit-tilasta: sen vaikutuksesta järjestelmän resursseja säästyy, koska tietokannan ei tarvitse varautua ROLLBACK-operaatioihin eikä monioperaatioisiin transaktioihin.

NTTM1-ajojen tuloksissa esiintyi epämääräistä vaihtelua. Tämän havainnollistamiseksi TM1:llä ja NTTM1:llä ajettiin toistuva sarja lyhyehköjä harjoitusajoja Standalone-1M -kokoonpanon kaikille konfiguraatioille. TM1-harjoitusten AQTh:t vaihtelivat eri ajokerroilla vain vähän, korkeintaan kaksi prosenttiyksikköä. NTTM1:llä keskimääräiset AQTh-arvot vaihtelivat huomattavasti enemmän. Ne esitetään suhteutettuna TM1:llä saatuihin tuloksiin taulukossa 4.4.

Taulukko 4.4: Neljän NTTM1-harjoitusajon AQTh:t suhteutettuna vastaavien TM1-harjoitusajojen tyypilliseen AQTh-arvoon.

Testikonfiguraatio	Harjoitusajo nro			
	1	2	3	4
Standalone-1M-10-70%	120 %	121 %	121 %	120 %
Standalone-1M-10-80%	123 %	122 %	123 %	121 %
Standalone-1M-10-90%	126 %	126 %	98 %	126 %
Standalone-1M-50-70%	117 %	124 %	115 %	120 %
Standalone-1M-50-80%	111 %	120 %	108 %	118 %
Standalone-1M-50-90%	110 %	121 %	108 %	120 %
Standalone-1M-100-70%	118 %	117 %	116 %	110 %
Standalone-1M-100-80%	121 %	121 %	119 %	114 %
Standalone-1M-100-90%	121 %	122 %	121 %	121 %

Taulukosta nähdään, että useimmat ajot sujuivat tasaisesti ja tässä testikokoonpanossa NTTM1 antoi noin 15-25 prosenttiyksikköä TM1:tä parempia tuloksia. Monilla vaakariveillä on silti myös yksi tai kaksi selvästi heikomman tuloksen saanutta ajoa. Ongelma havaittiin myös heikkoja tuloksia saaneiden ajojen aikana mitatuissa verkkoliikenteen seurantatilastoissa: verkon kuormitus oli alussa normaalitasolla, mutta laski koko ajon ajan hitaasti. Syytä tähän NTTM1:n ongelmaan ei saatu selville. Ongelman vaikutusta tuloksiin päätettiin pyrkiä vähentämään ajamalla tehokkuustesteissä jokaiselle testikonfiguraatiolle yhden ajon sijasta useita hieman lyhyempiä ajoja ja ottamalla niiden tuloksista keskiarvo. Ongelmattomien ajojen aikana testin kuormitus vakaantuu nopeasti, eikä ajoajan muutoksella ole juurikaan vaikutusta tuloksiin, kunhan testin pituus ylittää tietyn raja-arvon (n. 5-10 minuuttia).

4.6 Testiajot

Tietokanta-alustoista oltiin kiinnostuneita kahdelta eri kannalta: toisaalta NTTM1-testillä mitattavista tehokkuusarvoista, toisaalta taas siitä, miten alustat toteuttavat korkean saatavuuden järjestelmien ominaisuuksia. Koska eri testikonfiguraatioita oli paljon, ei tehokkuutta mitattaessa ollut aikaa ajaa niin pitkiä testejä, että esimerkiksi järjestelmän vakaudesta voitaisiin tehdä johtopäätöksiä. Tämän vuoksi testit on jaettu kahteen kategoriaan, jotka esitellään seuraavissa kohdissa.

4.6.1 Tehokkuustestit

Jokaiselle testikonfiguraatiolle ajettiin tehokkuustesti. Kukin testeistä koostui viidestä NTTM1-ajosta, jolloin testiajoja ajettiin yhteensä 540. Jokainen ajo sisälsi 5 minuutin kiihdytysvaiheen ja 15 minuutin varsinaisen testivaiheen.

Ajoista tallennettiin NTTM1:n tulostietokantaan keräämien mittausten lisäksi kullekin ajolle laskettu AQTh sekä kunkin testin ajojen AQTh-lukujen keskiarvo.

4.6.2 Saatavuustestit

Saatavuuden arvioinnista oltiin kiinnostuneita vain klusterialustojen kohdalla, eikä silloin ollut tarvetta vertailla eri konfiguraatioiden suorituskykyä. Sen vuoksi kaikki

saatavuustestit ajettiin korkeasti kuormitetulle kolmen koneen klusterille. Siihen käytettiin vain testikonfiguraatiota MCluster3-5M-100-80%. Saatavuustestit olivat seuraavat:

Vakaustesti. Järjestelmän toimintakyvyn säilymistä mitattiin pitkäkestoisen kuormituksen aikana. Sen mittaamiseksi ajettiin vastaavaa NTTM1-testiä kuin tehokkuustesteissäkin. Erona oli se, että 5 minuutin kiihdytysvaihetta seuraava varsinainen testivaihe kesti 15 minuutin sijasta 48 tuntia.

Solmunpudotustesti. Klusterin solmun (tai sen verkkoyhteyksien) kaatumista simuloitiin kytkemällä yhdestä palvelinkoneesta verkkokaapelit hetkeksi irti. Verkkokytkentöjen korjaamisen jälkeen seurattiin klusterin palautumista toimintakykyiseksi. Testin aikana oli käynnissä jatkuva NTTM1-kuormitus. Ne asiakasyhteydet, joita klusterista putoava solmu parhaillaan palvelee, katkeavat verkkoyhteyden katketessa. Todellinen tietokantaa käyttävä sovellus tyypillisesti yrittää tällöin avata yhteyden uudelleen, jolloin eri solmu ottaa pyynnön vastaan, ja toiminta voi jatkua kuin mitään ei olisi tapahtunut. NTTM1 ei tätä osaa, vaan se se tulkitsee testin epäonnistuneeksi ja keskeyttää sen, jos yksikin asiakasprosessien avaama yhteys katkeaa. NTTM1:n ongelma kierrettiin ajamalla solmunpudotustestissä peräkkäin jatkuvalla syötöllä lyhyitä testiajoja. Tällöin se ajo, jonka aikana solmu pudotetaan ja osa yhteyksistä katkeaa, jäisi ainoaksi testin vaiheeksi, josta tuloksia ei tallennu. Ajot olivat minuutin mittaisia ilman kiihdytysvaihetta, koska se on NTTM1-ajon minimipituus. Lisäksi testiohjelman lähdekoodia muokattiin niin, ettei se yhteyttä avatessaan kuluttanut aikaa tietokannan sisällön rakenteen tai koon tarkistamiseen.

Luku 5

Tulokset

5.1 Mittaustulokset

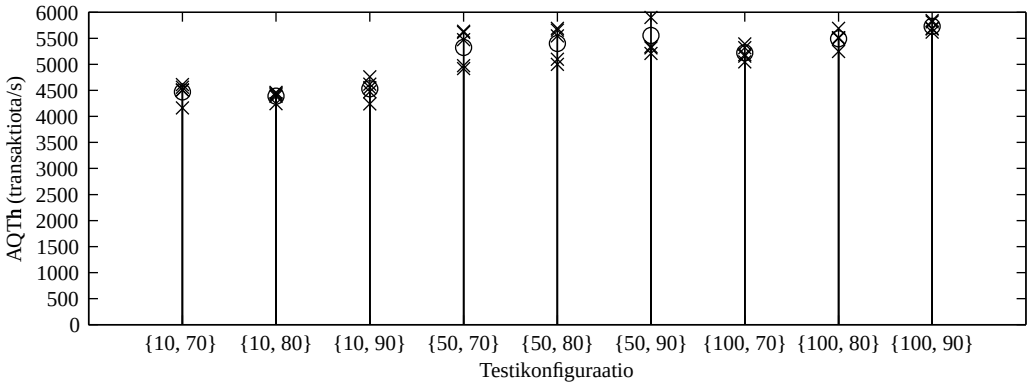
5.1.1 Tehokkuustestit

Seuraavilla sivuilla esitetään kustakin NTTM1-ajosta mitattu AQTh sekä kullekin testikonfiguraatiolle tehtyjen viiden ajon AQTh-lukujen keskiarvo. Tästä keskiarvosta käytetään jatkossa nimitystä *testin tulos*. Kullekin viiden luvun ryhmälle on laskettu myös variaatiokerroin havainnollistamaan saman konfiguraation ajokertojen keskiarvojen suhteellista hajontaa. Variaatiokerroin osoittaa, montako prosenttia luvut keskimäärin poikkeavat tuloksesta.

Kukin taulukoista 5.1-5.12 esittää yhden testikokoonpanon tulokset. Tilan säästämiseksi kukin testikokoonpanoon kuuluva testikonfiguraatio on nimetty asiakasprosessien lukumäärästä sekä luku- ja kirjoitussuhteesta koostuvalla lukuparilla {AP, LKS}. Omissa sarakkeissaan esitetään yksittäisistä ajoista mitatut luvut (AQTh 1 - AQTh 5), variaatiokerroin sekä tulos. Taulukoiden alla AQTh-luvut ja tulokset esitetään graafisesti siten, että kullekin testikonfiguraatiolle on oma sarakkeensa, jossa tulos on merkitty ympyrällä ja AQTh-luvut rastein.

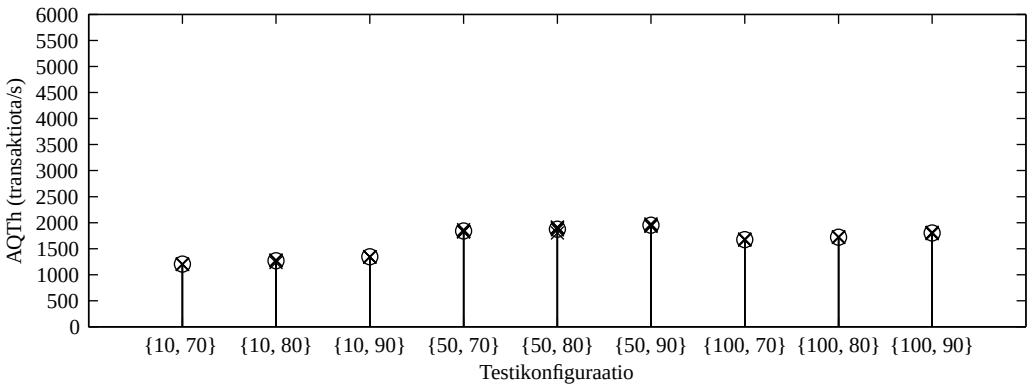
Taulukko 5.1: Testikokoonpanon Standalone-100k tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	4612	4506	4162	4506	4558	4,0 %	4469
{10, 80}	4387	4242	4439	4415	4464	2,0 %	4389
{10, 90}	4474	4550	4618	4239	4760	4,3 %	4528
{50, 70}	5475	4920	4974	5637	5613	6,6 %	5324
{50, 80}	5546	5001	5099	5656	5692	6,0 %	5399
{50, 90}	5901	5210	5313	6005	5339	6,7 %	5554
{100, 70}	5319	5157	5045	5391	5186	2,6 %	5220
{100, 80}	5249	5511	5502	5690	5506	2,9 %	5492
{100, 90}	5677	5816	5693	5619	5842	1,7 %	5729



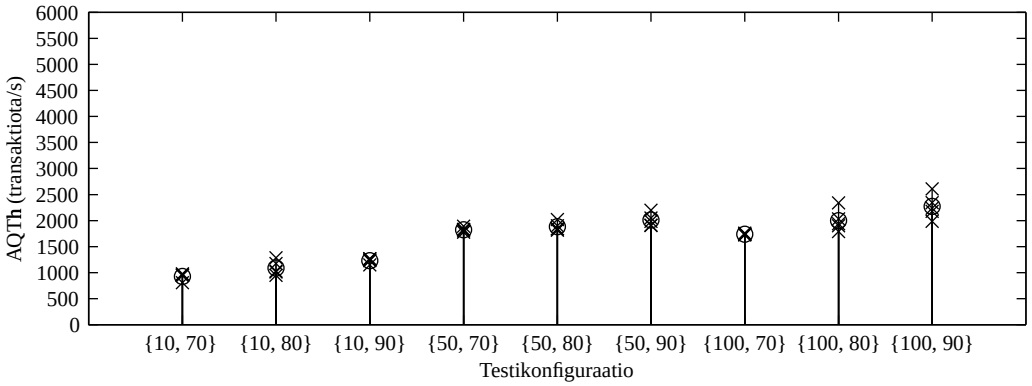
Taulukko 5.2: Testikokoonpanon MCluster1-100k tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	1195	1204	1205	1205	1206	0,4 %	1203
{10, 80}	1274	1235	1281	1270	1279	1,5 %	1268
{10, 90}	1337	1344	1344	1346	1344	0,2 %	1343
{50, 70}	1837	1837	1825	1822	1868	1,0 %	1838
{50, 80}	1802	1919	1885	1863	1910	2,5 %	1876
{50, 90}	1940	1975	1928	1951	1957	0,9 %	1950
{100, 70}	1663	1686	1669	1673	1679	0,5 %	1674
{100, 80}	1717	1723	1720	1724	1731	0,3 %	1723
{100, 90}	1810	1814	1795	1807	1790	0,6 %	1803



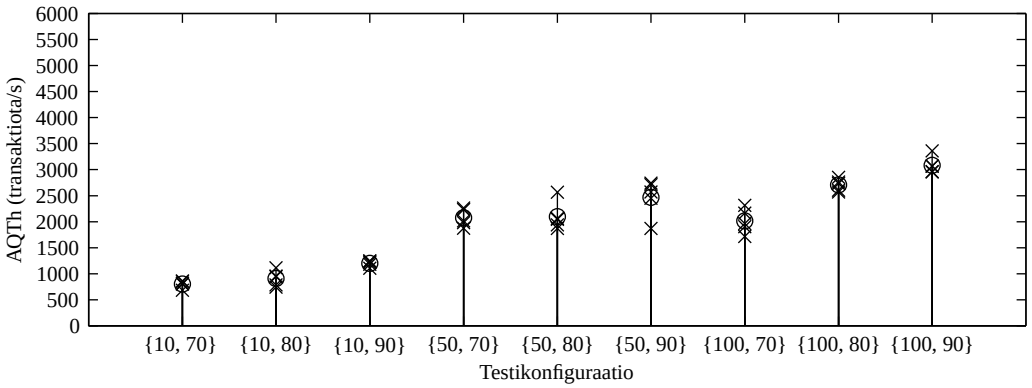
Taulukko 5.3: Testikokoonpanon MCluster2-100k tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	976	807	951	946	958	7,4 %	928
{10, 80}	1286	946	1028	1173	1011	12,7 %	1089
{10, 90}	1149	1241	1266	1230	1268	4,0 %	1231
{50, 70}	1822	1893	1830	1777	1802	2,4 %	1825
{50, 80}	1843	2022	1813	1899	1840	4,4 %	1883
{50, 90}	1911	2050	1903	1987	2201	6,1 %	2010
{100, 70}	1744	1730	1728	1760	1738	0,7 %	1740
{100, 80}	1899	1941	2340	2032	1788	10,5 %	2000
{100, 90}	2177	2240	2607	2343	1987	10,1 %	2271



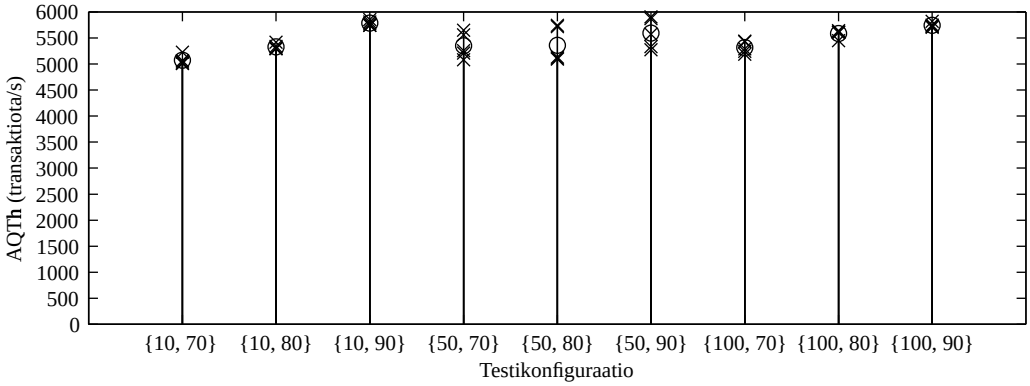
Taulukko 5.4: Testikokoonpanon MCluster3-100k tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	813	847	686	801	863	8,7 %	802
{10, 80}	786	744	943	1116	956	16,4 %	909
{10, 90}	1249	1220	1242	1183	1106	4,8 %	1200
{50, 70}	1875	2234	2017	2262	1984	8,1 %	2074
{50, 80}	2566	1941	2063	1870	2044	13,1 %	2097
{50, 90}	2569	2713	1868	2742	2453	14,4 %	2469
{100, 70}	2314	1905	1965	1719	2166	11,5 %	2014
{100, 80}	2573	2732	2854	2761	2606	4,3 %	2705
{100, 90}	2963	3066	3361	2955	3070	5,3 %	3083



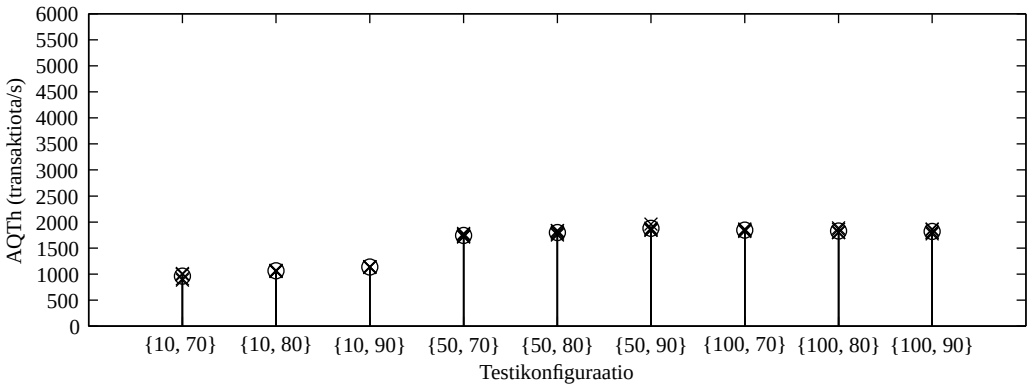
Taulukko 5.5: Testikokoonpanon Standalone-1M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	5224	5016	5038	5063	5009	1,7 %	5070
{10, 80}	5416	5339	5306	5292	5295	1,0 %	5330
{10, 90}	5853	5774	5740	5806	5742	0,8 %	5783
{50, 70}	5649	5207	5558	5257	5078	4,5 %	5350
{50, 80}	5741	5110	5714	5133	5092	6,3 %	5358
{50, 90}	5569	5333	5871	5274	5906	5,3 %	5591
{100, 70}	5440	5242	5424	5288	5187	2,1 %	5316
{100, 80}	5449	5602	5612	5616	5641	1,4 %	5584
{100, 90}	5745	5706	5818	5725	5725	0,8 %	5744



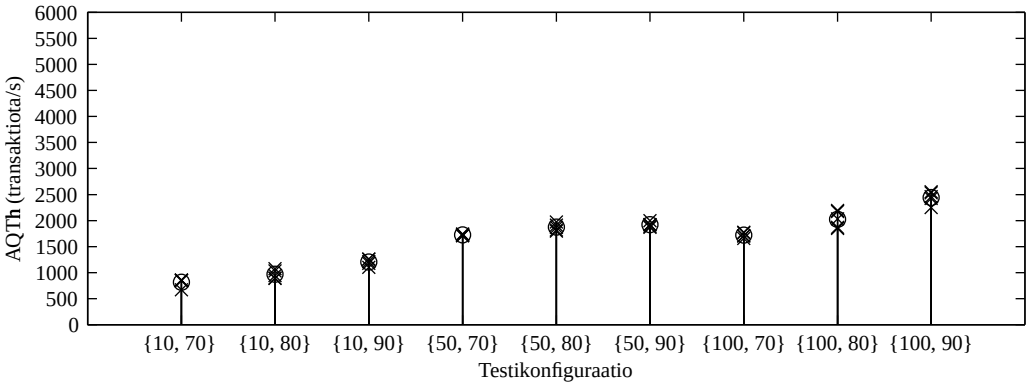
Taulukko 5.6: Testikokoonpanon MCluster1-1M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	996	886	1000	909	1011	6,0 %	960
{10, 80}	1054	1069	1053	1074	1062	0,9 %	1062
{10, 90}	1138	1138	1126	1134	1143	0,6 %	1136
{50, 70}	1781	1715	1723	1740	1765	1,6 %	1745
{50, 80}	1837	1802	1752	1771	1834	2,1 %	1799
{50, 90}	1849	1840	1867	1883	1958	2,5 %	1879
{100, 70}	1848	1836	1847	1867	1819	1,0 %	1843
{100, 80}	1806	1889	1856	1800	1798	2,2 %	1830
{100, 90}	1854	1801	1865	1806	1771	2,2 %	1820



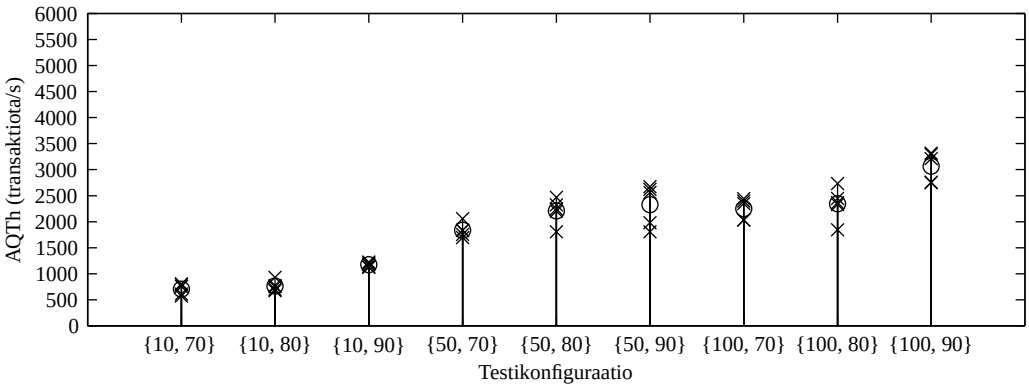
Taulukko 5.7: Testikokoonpanon MCluster2-1M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	858	857	671	846	856	10,1 %	818
{10, 80}	889	943	1035	897	1077	8,7 %	968
{10, 90}	1180	1102	1258	1259	1213	5,4 %	1202
{50, 70}	1708	1742	1721	1711	1735	0,9 %	1723
{50, 80}	1873	1797	1973	1818	1913	3,8 %	1875
{50, 90}	1871	1923	2001	1887	1923	2,6 %	1921
{100, 70}	1706	1765	1708	1774	1658	2,8 %	1722
{100, 80}	2175	2194	1849	2033	1871	8,0 %	2024
{100, 90}	2553	2458	2253	2531	2426	4,9 %	2444



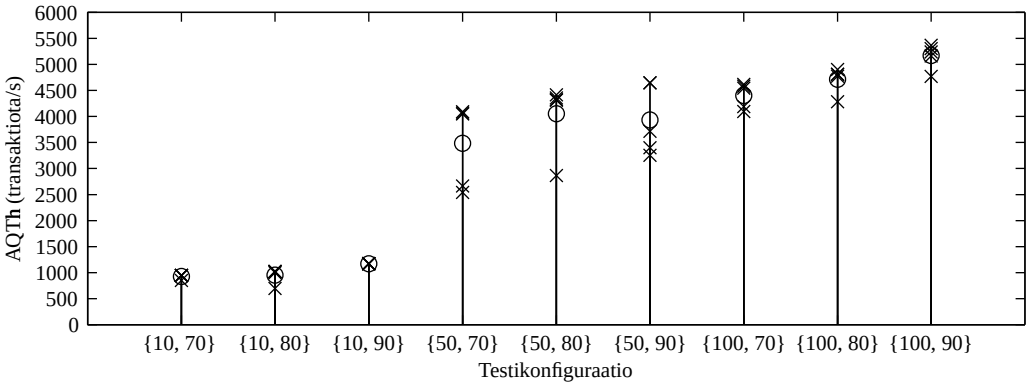
Taulukko 5.8: Testikokoonpanon MCluster3-1M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	756	806	787	571	611	15,2 %	706
{10, 80}	754	676	700	932	772	13,1 %	767
{10, 90}	1174	1124	1203	1137	1227	3,7 %	1173
{50, 70}	1697	1833	1753	1837	2057	7,5 %	1836
{50, 80}	2466	2324	2202	1806	2247	11,2 %	2209
{50, 90}	1809	2621	2560	2671	1983	17,3 %	2329
{100, 70}	2440	2033	2351	2393	2033	8,9 %	2250
{100, 80}	1845	2732	2331	2449	2365	13,7 %	2344
{100, 90}	3317	2748	3293	3210	2759	9,4 %	3065



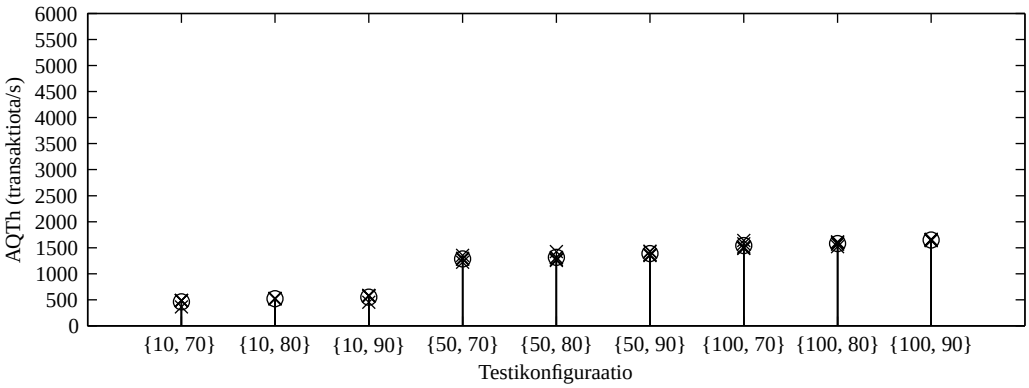
Taulukko 5.9: Testikokoonpanon Standalone-5M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	852	949	947	945	944	4,6 %	927
{10, 80}	999	1022	1021	1023	695	15,1 %	952
{10, 90}	1176	1169	1174	1170	1170	0,3 %	1172
{50, 70}	2665	4092	4066	2542	4049	23,1 %	3483
{50, 80}	4417	4311	2864	4356	4322	16,4 %	4054
{50, 90}	3717	3399	4645	4644	3255	17,1 %	3932
{100, 70}	4186	4614	4579	4544	4094	5,5 %	4404
{100, 80}	4811	4280	4806	4775	4901	5,2 %	4715
{100, 90}	5304	4771	5237	5164	5367	4,5 %	5169



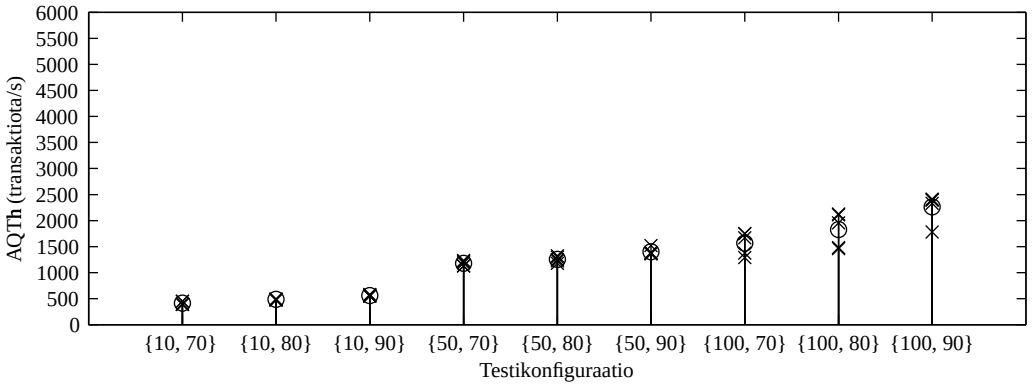
Taulukko 5.10: Testikokoonpanon MCluster1-5M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	491	484	483	484	364	11,8 %	461
{10, 80}	526	521	523	524	526	0,4 %	524
{10, 90}	586	571	453	575	582	10,2 %	553
{50, 70}	1352	1282	1222	1299	1282	3,6 %	1288
{50, 80}	1316	1254	1429	1308	1273	5,2 %	1316
{50, 90}	1351	1368	1362	1435	1423	2,8 %	1388
{100, 70}	1576	1516	1495	1639	1486	4,2 %	1542
{100, 80}	1523	1569	1598	1589	1615	2,2 %	1579
{100, 90}	1636	1645	1655	1657	1661	0,6 %	1651



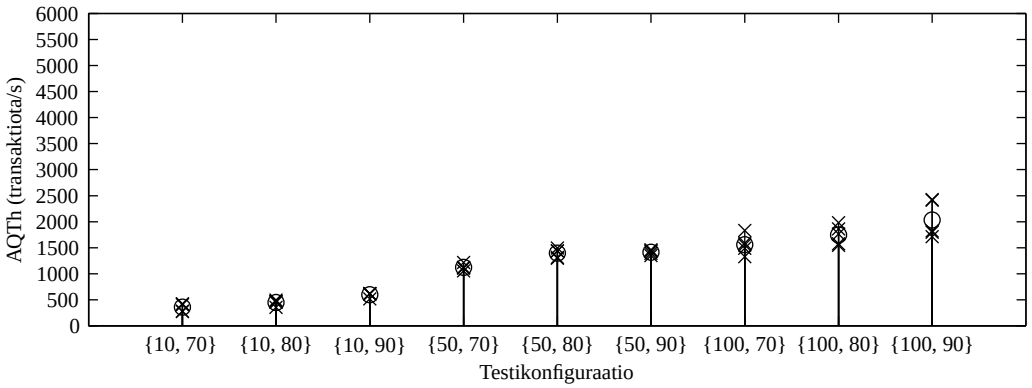
Taulukko 5.11: Testikokoonpanon MCluster2-5M tehokkuustestin tulokset.

{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	396	455	446	390	395	7,6 %	416
{10, 80}	489	483	473	492	489	1,6 %	485
{10, 90}	556	570	579	550	550	2,3 %	561
{50, 70}	1186	1236	1129	1212	1132	4,0 %	1179
{50, 80}	1227	1181	1297	1323	1250	4,5 %	1256
{50, 90}	1390	1521	1372	1366	1360	4,8 %	1402
{100, 70}	1377	1746	1291	1670	1750	13,8 %	1567
{100, 80}	1460	1957	1482	2125	2111	18,1 %	1827
{100, 90}	2398	2411	1779	2397	2331	12,0 %	2263

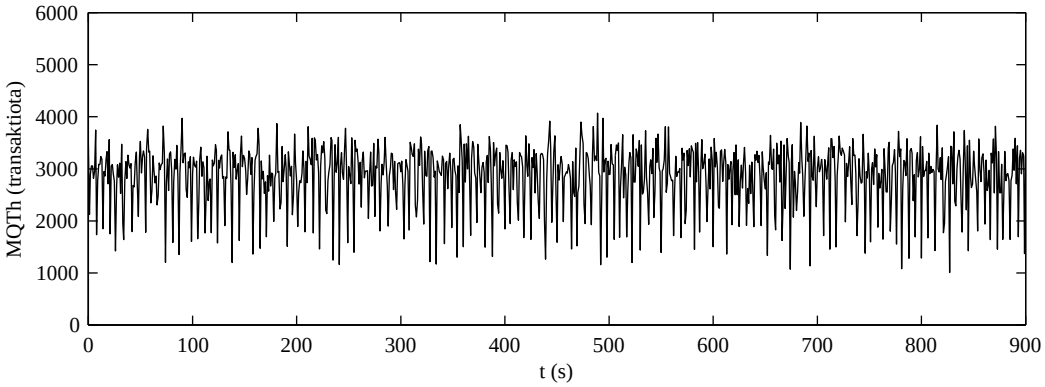


Taulukko 5.12: Testikokoonpanon MCluster3-5M tehokkuustestin tulokset.

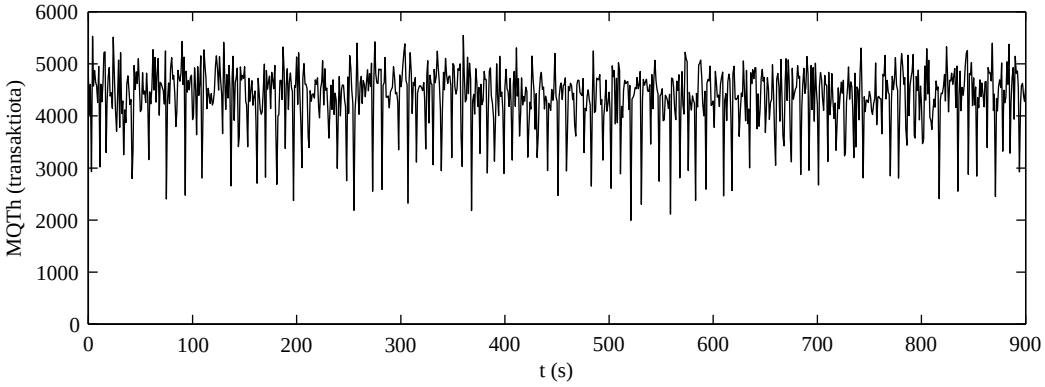
{AP, LKS}	AQTh 1	AQTh 2	AQTh 3	AQTh 4	AQTh 5	Vkerroin	Tulos
{10, 70}	274	422	409	418	285	20,7 %	361
{10, 80}	471	466	497	453	356	12,1 %	449
{10, 90}	620	611	618	618	515	7,6 %	596
{50, 70}	1113	1055	1123	1217	1114	5,2 %	1124
{50, 80}	1439	1441	1299	1317	1495	6,1 %	1398
{50, 90}	1350	1435	1439	1390	1462	3,1 %	1415
{100, 70}	1589	1490	1556	1832	1330	11,7 %	1559
{100, 80}	1766	1863	1979	1577	1545	10,6 %	1746
{100, 90}	1796	2427	1712	2411	1824	17,4 %	2034



Tehokkuustestien tulosten kuvaajista on nähtävissä, että osassa testeistä tuloksissa esiintyi huomattavaa vaihtelua. Yksittäisen ajon suoritusta voidaan tarkastella lähemmin kuvaajasta, joka esittää MQTh-arvon muutoksia testin aikana. Suurin AQTh:n vaihteluväli oli testikonfiguraatiolla Standalone-5M-50-80%, jossa korkein AQTh oli 54 % suurempi kuin saman konfiguraation alin AQTh. Kyseisten ajojen MQTh-kuvaajat ovat kuvissa 5.1 ja 5.2.



Kuva 5.1: Heikoin konfiguraatiolla Standalone-5M-50-80% mitattu ajo (AQTh 2864).



Kuva 5.2: Paras konfiguraatiolla Standalone-5M-50-80% mitattu ajo (AQTh 4417).

Lyhyellä aikavälillä tarkasteltuna MQTh heittelee rajusti ollen hankalasti ennakoitavissa, mutta jo minuutin kestäneen mittauksen perusteella voidaan yleensä arvioida lukema, jonka tuntumassa keskiarvo pysyy ajon loppuun saakka. Kaikki tehokkuustestien ajot tuottivat visuaalisesti samankaltaisia käyriä, joissa ainoa havaittava ero on MQTh:n vaihtelun keskittyminen eri AMQTh:n ympärille. Koska kaikkien ajojen

AQTh on jo esitetty edellä, jätetään loppujen 538 ajon MQTh-kuvaajat tässä esittämättä niiden tuoman vähäisen lisäinformaation vuoksi.

5.1.2 Saatavuustestit

Vakaustestissä hyväksyttyyn suoritukseen riitti se, että pitkä testiajo pysyi toiminnassa loppuun asti ilman virhetilanteita. Tulos esitetään taulukossa 5.13.

Taulukko 5.13: Vakaustestin tulos.

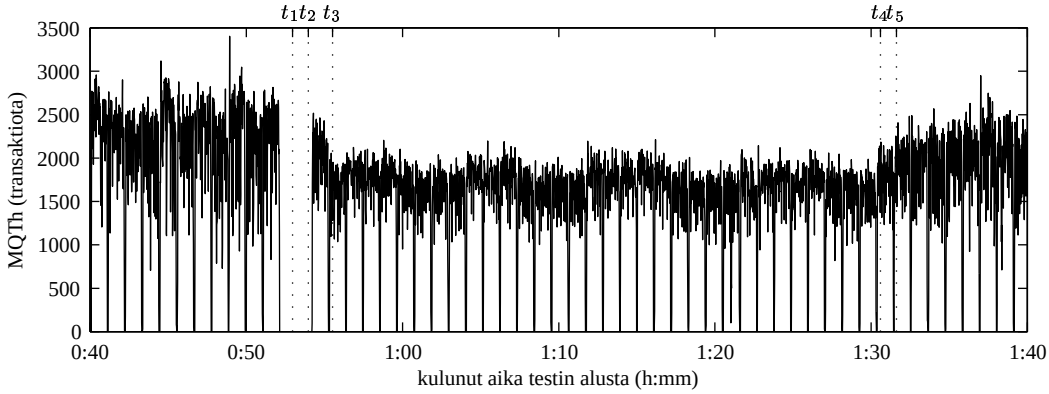
Testikonfiguraatio	Testin kesto (h)	Tulos
MCluster3-5M-100-80%	48	Hyväksytty

Solmunpudotustestin MQTh-kuvaaja on kuvassa 5.3. Siinä esitetään solmun pudottamista ja siitä toipumista ympäröivä tunnin mittainen ajanjakso. Kuvaajaan merkittyjä testin kannalta oleellisia ajanhetkiä kuvataan tarkemmin taulukossa 5.14.

NTTM1:n lähdekoodiin tehdyillä muutoksillakaan tietokannan kuormitusta ei saatu aivan jatkuvaksi. Jokaisen minuutin mittaisen testiajon väliin jäi noin 2 sekunnin tauko. Tämä näkyy kuvaajassa siten, että MQTh käy noin minuutin väliajoin nollassa. Tauot aiheuttivat sen verran säännöllistä hukka-aikaa, että kymmeneen minuuttiin mahtui vain reilut yhdeksän ajoa.

5.2 Mittaustulosten arviointi

Kuormitustestin hyväksytty tulos ja ylipäätään kaikkien ajojen onnistuminen osoitti tämän työn puitteissa vakaan M/Cluster-ympäristön rakentamisen olevan mahdollista. Myös solmunpudotustesti sujui onnistuneesti. Sitä, että pudotettavaan solmuun parhaillaan auki olleet asiakasyhteydet katkesivat, voidaan pitää hyväksyttävänä. Solmun uudelleenliittymiseen kulunut aika oli pitkä, mutta se sujui silti huomattavasti nopeammin kuin vaikkapa viiden miljoonan asiakkaan tietojen populoiminen uudelleen yhden koneen MySQL-tietokantaan. Korkeaa saatavuutta enemmän haittaava ominaisuus on se, että solmu ei palaa klusteriin automaattisesti, vaan ylläpitäjän on tehtävä se käsin. Toki yleensä on järkevää, ettei klusteri tee järjestelmän tehokkuutta heikentäviä ratkaisuja ilman ihmisen kuittausta ja valvontaa.



Kuva 5.3: Solmunpudotustestiajo.

Taulukko 5.14: Solmunpudotustestin vaiheet ja oleelliset ajanhetket.

Ajan-hetki	Aika testin alusta	Tapahtuma
t_0	0:00:00	Testiajojen sarja käynnistyy kolmisolmuisessa klusterissa (solmut A, B ja C).
t_1	0:53:00	C:n verkkoyhteys katkaistaan. Osan asiakasprosesseista yhteys katkeaa. Parhaillaan käynnissä oleva testiajo tulkitsee testin epäonnistuneeksi eikä tallenna tuloksia.
t_2	0:54:00	Verkkoyhteys kytketään takaisin. C alkaa palauttaa yhteyksiään muihin solmuihin. Hieman tämän jälkeen uusi testiajo käynnistyy ja asiakasprosessit saavat yhteyden solmujen A ja B muodostamaan kahden koneen klusteriin.
t_3	0:55:31	A lopettaa asiakasprosessien palvelemisen ja aloittaa tietokannan taulujen kopioimisen C:lle. Asiakasprosesseja palvelee enää vain B.
t_4	1:30:36	C saa oman replikansa ajan tasalle ja aktivoituu.
t_5	1:31:37	A aktivoituu ja klusteri on jälleen täysilukuinen.

Seuraavassa tehokkuustestien tuloksia analysoidaan tarkastelemalla yksi kerrallaan kutakin testiparametria ja sen vaikutuksia tuloksiin.

Asiakasprosessien määrä. 10 asiakasprosessilla saatiin selvästi heikoimpia tuloksia, mikä merkitsee sitä ettei pieni määrä asiakasprosesseja riittänyt kuormittamaan tietokantajärjestelmää täydellä teholla. 100 prosessia sai pääsääntöisesti parhaat tulokset. Koska kuitenkin 50 ja 100 asiakasprosessin tuloserot olivat keskenään pieniä, oli useimpien testikokoonpanojen täyteen kuormittamiseen vaadittava prosessimäärä 10 ja 50 kappaleen välillä.

Populaation koko. Tietokannan koon kasvatus laski useimmiten tehokkuutta. 100k- ja 1M-populaatioiden väliset erot eivät olleet suuria. Useissa konfiguraatioissa 1M-populaatiolla saavutettiin jopa 100k:ta parempia tuloksia, varsinkin silloin kun asiakasprosesseja oli 100. Selvempi ero havaitaan 1M- ja 5M-populaatioiden välillä. Tällä tarkasteluvälillä 10 asiakasprosessin tehokkuudessa oli hyvin huomattava lasku. 10 asiakasprosessin ja 5 miljoonan tilaajapopulaation yhdistelmä tuottikin testien heikoimmat tulokset.

Luku- ja kirjoitusoperaatioiden suhteellinen määrä. Lukusuhde vaikutti tuloksiin oletetusti, eli sen suurentaminen paransi tuloksia. Tämäkään ei silti aina pätenyt, sillä esimerkiksi kokoonpanossa MCluster1-1M lukuoperaatioiden lisäämisellä oli päinvastainen vaikutus silloin, kun asiakasprosesseja oli 100. Tämän aiheuttajaksi voidaan silti perustella NTTM1-tulosten yleinen satunnaisuus, sillä näiden 100 asiakasprosessin ajojen tulosten erot olivat hyvin pieniä.

Solmujen lukumäärä. Solmujen lukumäärän vaikutus tuloksiin vaihteli merkittävästi konfiguraatioista riippuen. Esimerkiksi rajaamalla tarkasteltaviksi vain yhden ja kolmen solmun konfiguraatiot, joissa asiakasprosesseja oli 100 ja luku- ja kirjoitusoperaatioiden suhde 80:20, vaihtelee järjestelmälle laskettava skaalauskerroin silti lukujen 0,37 ja 0,52 välillä. Niissä tapauksissa, joissa solmujen lisääminen heikensi suorituskykyä, ei skaalauskertoimen laskeminen ole edes järkevää. Tämän vuoksi solmujen lukumäärien eroja tyydytään tarkastelemaan lyhyesti asiakasprosessien määrän mukaan ryhmiteltynä:

- 10 asiakasprosessia: Kokoonpanoilla ei ollut juurikaan eroja, mutta yhden solmun klusteri oli usein tehokkain.
- 50 asiakasprosessia: Yhden ja kahden solmun erot olivat pieniä, mutta kolmen solmun kokoonpano sai selvästi parhaat tulokset.

- 100 asiakasprosessia: Kuten 50 asiakasprosessillakin, kolmen solmun kokoonpano sai paremmat tulokset kuin yhden ja kahden solmun kokoonpanot. Poikkeuksen kuitenkin muodosti 5 miljoonan populaatio, jolla kahden solmun klusteri oli kolmen solmun klusteria tehokkaampi.

Tietokanta-alusta. MySQL sai tehokkuustesteissä selkeästi korkeampia arvoja kuin M/Cluster-kokoonpanot. Kolmen solmun klusteri saavutti parhaimmillaan vain puolet yksittäisen MySQL-palvelimen tehokkuudesta.

5.3 Johtopäätökset

Testaustyön päämääränä oli arvioida, miten testialustat täyttävät HLR-sovelluksen tarpeet. Seuraavat arviot perustuvat saatavilla olleen dokumentaation, työn toteutuksen aikana saadun kokemuksen sekä mittaustulosten muodostamaan yleiskuvaan tuotteiden ominaisuuksista ja suorituskyvystä.

MySQL Cluster. Tuotteen epävakaus testiympäristössä antoi siitä viimeistelemättömän vaikutelman ja esti saatavuusominaisuuksien ja suorituskyvyn mittaamisen. Tietokannan säilyttäminen pelkässä keskusmuistissa rajoittaa liikaa tietokannan kokoa ja voi laajassa järjestelmässä olla ongelma. Datasolmun lisäämisen työläyden vuoksi tuotteen skaalautumiskyky on tähän käyttötarkoitukseen riittämätön. MySQL Cluster on kuitenkin jatkuvan kehitystyön alla, joten tässä työssä testattua uudemmat ohjelmaversiot saattavat tuoda korjauksia mainittuihin ongelmiin. Tuotteen hyödyllisyydestä luo lupauksia NDB-moottorin tuki transaktioille, mikä on käytännön välttämättömyys useimpien tietokantasovellusten toiminnalle.

Emic M/Cluster. M/Cluster suoriutui monilla testien osa-alueilla päinvastaisesti kuin MySQL Cluster. Testit saatiin suoritettua vakaasti ja ilman ongelmia. Selkeä arkkitehtuuri, jossa klusterin solmujen alustana toimi paikallinen MySQL, säästi monilta keinotekoisen oloisilta rajoitteilta, kuten MySQL Clusterin tietokantataulujen pakotettu muistinvaraisuus. Transaktiotuen puute oli kuitenkin niin kriittinen rajoite, että yksinään sen vuoksi Emic M/Clusteria ei voida pitää HLR-käyttöön sopivana alustana. Toinen sen käyttökelpoisuutta kyseenalaistava seikka on selvästi heikompi suoriutuminen tehokkuustesteistä yhden koneen MySQL:ään verrattuna. Klusteri ei saavuttanut MySQL:n suorituskykyä, vaikka solmuja oli käytössä kolme. Solmunpudotustestin tuloksissa on huomattavaa, että tehokkuusero tilanteissa, joissa toimivia solmuja oli kolme tai yksi, oli pieni. Tästä voidaan päätellä, että klusteroinnista saatavat edut

keskittyvät järjestelmän vikasietoisuuden ja toipumiskyvyn parantamiseen. Nämä parannukset joudutaan tekemään tehokkuuden kustannuksella. Jos M/Clusterin tuleviin versioihin saadaan mukaan toimiva transaktiotuki, ja se saadaan testattua ilman testiohjelman muokkaamista M/Clusterille suotuisammaksi, näiden johtopäätösten arviointi uudelleen on aiheellista.

Jos klusteroinnin mahdollistamat parannukset saatavuuteen ovat välttämättömiä palvelun toiminnalle, ei puhdas tehokkuusvertailu klusteroimattoman ja klusteroidun järjestelmän ole kuitenkaan järkevää. Klusterointia ei kannattaisi käyttää lainkaan tehoa heikentämässä, jos sen tuomia parannuksia ei tarvita. Vaikka yhden koneen MySQL oli huomattavasti tehokkaampi kuin M/Cluster, se ei yksinään tarjoa mitään vertailukelpoisia keinoja vikasietoisuuden tai saatavuuden parantamiseen.

Testien toteutuksen ja tulosten koostamisen aikana pantiin merkille eräitä seikkoja, joita pidettiin tarpeellisina mainita tulosten julkaisun yhteydessä. Seuraavien huomautusten on tarkoitus tuoda paremmin esiin se, että testattujen tuotteiden ominaisuuksista tai yleisestä sopivuudesta telekommunikaatiosovelluskäyttöön ei voida tehdä lopullisia johtopäätöksiä tämän työn tulosten perusteella.

Tässä työssä esitetyt luvut kertovat testattujen sovellusten tehokkuudesta ainoastaan tässä laitteisto- ja ohjelmistoympäristössä. Ne eivät ole suoraan vertailtavissa muissa ympäristöissä mitattuihin tuloksiin eikä niiden perustella voida arvioida millaisia tuloksia olisi saatu, jos työssä käytetty laitteisto olisi ollut erilainen.

Testiympäristön palvelinkoneina toimineet PC:t olivat malliltaan uusia ja edustivat laskenta- ja muistikapasiteetiltaan kuluttajahintaisten työasemien sen hetkistä kärkipäätä. Valinta tehtiin, koska haluttiin nähdä, millainen tehokkuus kuluttajahintaisilla koneilla voidaan saavuttaa. Koneiden tehosta oli kuitenkin myös haittaa, sillä se hankaloitti niiden kuormittamista täydellä kapasiteetilla. Pienempitehoisilla ja -muistisilla koneilla mitatut tulokset olisivat olleet yhtä vertailukelpoisia keskenään kuin tässä työssä saadut, mutta olisi mahdollisesti ollut helpompaa löytää testiparametrien arvoja, joiden ylittyessä järjestelmän käyttäytyminen muuttuu selvästi. Pidempään markkinoilla olleiden laitteiden Linux-ajurituki on myös yleensä vakaampi. Sen ansiosta myös MySQL Cluster olisi voinut olla riittävän vakaa testattavaksi. Solmujen määrän vaikutusta suorituskykyyn olisi voitu mitata luotettavammin käyttämällä useampaa kuin kolmea solmua. Niiden määrää kuitenkin rajoittivat käytettävissä olevat resurssit, ja skaalautuvuuden kattavampi tutkiminen olisi jo erillisen tutkimuksen aihe.

Tuotantokäytössä järjestelmän sovelluskohtaista tehokkuutta yritetään yleensä parantaa optimoinnilla. Se voi olla pitkä prosessi, jossa palvelun toimintaa säädetään halutuun suuntaan muokkaamalla tietokantasovelluksen, käyttöjärjestelmän ja laitteiston asetuksia. Muutosten vaikutukset saattavat näkyä vasta pitkäaikaisessa käytössä, ja pikkutarkkojen säätöjen tekeminen voi vaatia pitkällistä kokemusta ja asiantuntemusta käytettävästä ohjelmistosta. Tämän kaltaisen kokemuksen puutteen vuoksi ja ajan säästämiseksi tässä työssä jätettiin optimointi tekemättä, ja tasapuolisuuden nimissä kieltäydyttiin vastaanottamasta tuotteiden valmistajilta apua tehokkuuteen vaikuttavien asetusten muokkaamiseen. Ohjelmien perusasetukset ovat tyypillisesti sellaisia, että niiden alaisuudessa järjestelmä on mahdollisimman varmatoiminen useimmissa erilaisissa käyttötarkoituksissa, tietysti tehokkuuden kustannuksella. Sovelluskohtaisen optimoinnin jälkeen tuotteet olisivat voineet antaa tehokkuustesteissä tässä esitetyistä poikkeavia tuloksia.

Luku 6

Yhteenveto

Ilmaisiin tai edullisiin ohjelmistoihin ja kuluttajahintaisiin laitteisiin perustuvia ratkaisuja hajautettujen tietokantaklustereiden rakentamiseen on saatavilla useita. Edullisuus tekee niistä kiinnostavan vaihtoehdon mm. telekommunikaatioverkon yhteyksien ja asiakastietojen hallintaan käytettävän HLR-rekisterin tietokantamoottoriksi. Siinä kriittisiä ominaisuuksia ovat palvelun katkoton toiminta, skaalautuvuus kasvavalle asiakasmäärälle ja tehokkuus raskaalla lukuoperaatiopainotteisella kuormituksella.

Tässä diplomityössä haluttiin selvittää tietokantatuotteiden käyttökelpoisuutta HLR:n alustana. Sitä varten rakennettiin testiympäristö, jossa eri tietokantajärjestelmiin kohdistettiin muokatulla TM1-testisovelluksella tuotettu testikuorma. Verrokkialustana oli yhden koneen MySQL-tietokanta. Varsinaisina testikohteina olivat MySQL-klusteroinnin mahdollistavat tuotteet MySQL Cluster ja Emic M/Cluster, joista tosin vain jälkimmäinen oli testiympäristössä riittävän vakaa toimiakseen testialustana. Tehokkuusmuutosten tarkastelemiseksi määriteltiin joukko testiparametreja, joiden eri yhdistelmillä testit ajettiin. Saatavuusominaisuuksia arvioitiin pitkäkestoisella kuormituksella sekä testillä, jossa mallinnettiin yhden palvelinkoneen hetkellistä vikatilannetta.

Muokatun TM1-testin tehokkuuden huomattiin olevan vaikeasti ennakoitavissa ja tulosten vaihtelevan epätoivotusti ajokerrasta riippuen. Sen vaikutusta pienennettiin laskemalla tulokseksi useiden samoilla parametreilla tehtyjen ajojen tulosten keskiarvo. Siitä huolimatta harva testiparametri osoitti selkeästi parantavaa tai heikentävää vaikutusta tehokkuuteen. Vaikutukset saattoivat olla riippuvaisia muiden parametrien arvoista. Selvää kuitenkin oli, että yhden koneen MySQL oli huomattavasti tehokkaam-

pi kuin M/Clusterilla toteutettu kolmen koneen klusteri. Saatavuustesteistä klusteri suoriutui hyväksytysti, ja voidaankin päätellä, että järjestelmän edut keskittyvät palvelun vikasietoisuuden ja luotettavuuden parantamiseen, valitettavasti tehokkuuden kustannuksella.

M/Clusterin transaktiotuen puute osoitti sen sopivan heikosti HLR-käyttöön. Myöskään klusteroinnin tuomista tehokkuusparannuksista ei saatu näyttöä. Saatavuusominaisuuksiltaan tuote oli lupaava. On myös huomattava, että klusterin asetuksia ei optimoitu tehokkuuden kannalta lainkaan, joten sen todellinen tuotantoympäristössä saavutettava nopeus tässä tehtävässä jäi selvittämättä. Mahdollisesti uudempiin versioihin kehitettävä transaktiotuki voi tehdä aiheelliseksi arvioida tässä työssä tehtyjä johtopäätöksiä uudelleen.

Lähdeluettelo

[1] Banga, G. et al. *Measuring the Capacity of a Web Server*. 1st USENIX Symposium on Internet Technologies and Systems (USITS 1997), Monterey, Dec. 8-11, 1997.

[2] Brewer, E. *Lessons from Giant-Scale Services*. IEEE Internet Computing 5, 4 (2001). s. 46-55.

[3] *C-JDBC: Clustered JDBC*. WWW, viitattu: tammikuu 2006.
`http://c-jdbc.objectweb.org/`

[4] Colton, M. *Benchmarking for Network Infrastructure Applications*. RTC Magazine 14, 2 (2005).

[5] *Debian: The Universal Operating System*. Software in the Public Interest, Inc. WWW, viitattu: tammikuu 2006.
`http://www.debian.org/`

[6] *Emic M/Cluster 2.5 Administrator's Guide, document issue 2.1*. Emic Networks. 2005.

[7] *Fedora Core*. Red Hat, Inc. WWW, viitattu: tammikuu 2006.
`http://fedora.redhat.com/`

[8] *GNU General Public License*. Free Software Foundation, Inc. 1991. WWW, viitattu: huhtikuu 2006.
`http://www.gnu.org/copyleft/gpl.html` Version 2 painos.

[9] Gray, J. et al. *The Dangers of Replication and a Solution*. Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996. ACM Press, 1996. s. 173-182.

[10] Jiménez-Peris, R. et al. *How to Select a Replication Protocol According to Scalability, Availability, and Communication Overhead*. 20th Symposium on Reliable Distributed Systems (SRDS 2001), New Orleans, Oct. 28-31, 2001. IEEE Computer Society, 2001. s. 24-33.

[11] Katz, E. et al. *A Scalable HTTP Server: The NCSA Prototype*. Computer Networks and ISDN Systems 27, 2 (1994). s. 155-164.

[12] *Lustre: scalable, secure, robust, highly-available cluster file system*. Cluster File Systems, Inc. WWW, viitattu: tammikuu 2006.
`http://www.lustre.org/`

[13] *M/Cluster*. Continuent. WWW, viitattu: helmikuu 2006.
`http://www.continuent.com/index.php?option=com_content&task=view&id=211&Itemid=168`

[14] Merkey, P. *Beowulf History*. WWW, viitattu: tammikuu 2006.
`http://www.beowulf.org/overview/history.html`

[15] *MySQL: The world's most popular open source database*. MySQL AB. WWW, viitattu: helmikuu 2006.
`http://www.mysql.com/`

[16] *MySQL 3.23, 4.0, 4.1 Reference Manual, revision 1030*. MySQL AB. 25.1.2006. WWW, viitattu: tammikuu 2006.
`http://dev.mysql.com/doc/refman/4.1/en/`

[17] *MySQL Cluster*. MySQL AB. WWW, viitattu: huhtikuu 2006.
`http://www.mysql.com/products/database/cluster/`

[18] *MySQL Connector/ODBC*. MySQL AB. WWW, viitattu: helmikuu 2006.
`http://www.mysql.com/products/connector/odbc/`

[19] *Network Database Benchmark*. Nokia Corporation & Helsinki Open Source Laboratory. WWW, viitattu: helmikuu 2006.
`http://hoslab.cs.helsinki.fi/homepages/ndbbenchmark/`

[20] *Open Database Connectivity (ODBC)*. Microsoft Corporation. WWW, viitattu: helmikuu 2006.
`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/dasdkodbcoverview.asp`

[21] *PostgreSQL: The world's most advanced open source database*. The PostgreSQL Global Development Group. WWW, viitattu: tammikuu 2006.
`http://www.postgresql.org/`

[22] *Red Hat Enterprise Linux: The corporate Linux standard*. Red Hat, Inc. WWW, viitattu: tammikuu 2006.
`http://www.redhat.com/software/rhel/`

[23] *Slony-I: A replication system for PostgreSQL*. WWW, viitattu: tammikuu 2006.
`http://gborg.postgresql.org/project/slony1/`

[24] *Solid Information Technology*. WWW, viitattu: maaliskuu 2006.
`http://www.solidtech.com/`

[25] Stonebraker, M. *The Case for Shared Nothing*. IEEE Database Engineering Bulletin 9, 1 (1986). s. 4-9.

[26] Strandell, T. *Open Source Database Systems: Systems study, Performance and Scalability*. Pro Gradu. Helsingin yliopisto, Tietojenkäsittelytieteen laitos. 2003.

[27] *Telecom One (TM1) Benchmark*. Solid Information Technology. WWW, viitattu: helmikuu 2006.
`http://www.solidtech.com/en/developers/CarrierGrade/tm1.asp`

[28] *Telecom One (TM1) Benchmark Description, version 1.0.7*. Solid Information Technology. 1.3.2005.

[29] *TPC-C Benchmark 5.6*. Transaction Processing Performance Council. WWW, viitattu: tammikuu 2006.
`http://www.tpc.org/tpcc/`