Haaga-Helia
ammattikorkeakoulu Oy

Vedonlyöntijärjestelmän laajentaminen vinttikoirakilpailuun

Jorma Tenni

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2019

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Jorma Tenni

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön nimi
Vedonlyöntijärjestelmän laajentaminen vinttikoirakilpailuihin

Sivu- ja liitesivumäärä
32+9

Opinnäytetyössä laajennettiin olemassa olevaa järjestelmää toimimaan vedonlyöntipörssin vinttikoirakilpailukohteissa. Vinttikoirakilpailut ovat vedonlyöntikohteina hyvin nopeatahtisia ja lyhytkestoisia. Tämän takia niissä kauppaa käyvien järjestelmien tulee olla suorituskyvyltä tehokkaita ja järjestelmän vasteaikojen on oltava lyhyitä.

Opinnäytetyö tehtiin vedonlyöntialalla toimivassa yrityksessä, joka kehittää kerroinasettelun optimointiin keskittyvää palvelua vedonvälitysyrityksille. Opinnäytetyössä haluttiin ymmärtää paremmin vedonlyöntipörssien vinttikoirakilpailukohteiden vaatimuksia, sekä soveltuisiko yrityksen nykyinen järjestelmä ja arkkitehtuuri sellaisenaan vedonlyöntipörssien nopeatahtisiin kohteisiin. Vinttikoirakilpailuita varten järjestelmään tehtävän laajennoksen lisäksi opinnäytetyössä mitattiin ja analysoitiin järjestelmän suorituskykyä, selvitettiin suorituskyvyn ongelmakohtia sekä parannettiin järjestelmän tehokkuutta korjaamalla mittauksessa havaittuja suorituskyvyn ongelmakohtia.

Opinnäytetyössä esitellään vedonlyöntiä ja vedonlyöntialaa, sekä käydään läpi yrityksen järjestelmää ja siihen tehtävää laajennosta. Laajennosta ja koko järjestelmän tehokkuutta analysoitiin mittaamalla järjestelmän vasteaikoja. Mittaustuloksia tutkimalla järjestelmästä valikoitui hitaita osioita ja löydettiin korjausehdotuksia. Korjaukset tekemällä, voitiin uuden mittauksen ja analysoinnin perusteella todeta järjestelmän toimivan huomattavasti tehokkaammin vasteaikojen pienentyessä murto-osaan lähtötilanteesta.

Asiasanat
Ohjelmistosuunnittelu, ohjelmistokehitys, analyysi, vedonlyönti

Sisällys

1 Johdanto

Vedonlyöntipörssit ovat vedonlyöntiin liittyviä sivustoja, joissa henkilöt ja yritykset voivat lyödä vetoa toisiaan vastaan. Vedonlyöntiä vedonlyöntipörsseissä voi myös automatisoida rakentamalla järjestelmiä, jotka käyvät lyövät vetoa sopivien algoritmien avulla. Tällaista kaupankäyntiä tekevän järjestelmän kannattavuus voi olla pienestä kiinni, hidastelu vetoa lyödessä voi merkitä kannattavuuden menetystä. Kohteissa joissa vetoa lyödään paljon ja lyhyen ajan sisällä, vaaditaan vetoa lyövältä järjestelmältä tehokkuutta ollakseen kannattava.

Opinnäytetyön aiheena on toteuttaa vedonlyöntialalla toimivan yrityksen kerroinpäivitysjärjestelmään laajennos, jonka avulla järjestelmä toimii vedonlyöntipörssin vinttikoirakilpailukohteissa. Laajennoksen avulla selvitetään paitsi toiminnan laajennusmahdollisuuksia näihin kohteisiin, myös yrityksen järjestelmän suorituskykyä nopeatahtisissa kohteissa, joissa vetotapahtumia tulee paljon lyhyen ajan sisällä.

Yrityksen tämänhetkinen palvelu tarjoaa vedonvälittäjille hinnoittelualgoritmia monipuolisesti vedonlyöntikohteisiin, jotka eivät ole vielä alkaneet (ns. pre-game kohteet). Vedonvälittäjät ovat yrityksiä, jotka tarjoavat asiakkailleen mahdollisuuden lyödä vetoa monenlaisista vedonlyöntitapahtumista. Yleisiä vedonlyöntikohteita ovat urheilutapahtumat, mutta vetoa voi lyödä myös esimerkiksi vaalituloksista tai Euroviisujen voittajasta. Vedonlyöntialalla kohteet voidaan jakaa ”pre-game” ja ”in-play”-kohteiksi. Pre-game kohteissa lyödään vetoa tapahtumasta, joka ei ole vielä alkanut. In-play kohteissa lyödään vetoa käynnissä olevasta ottelusta (tai tapahtumasta). Opinnäytetyössä käytettäviä vinttikoirakilpailuja pelataan vedonlyöntipörsseissä pre-game kohteina.

Pre-game kohteissa voittajan todennäköisyydet vaihtuvat yleensä verkkaisesti, koska kohteeseen vaikuttavia tapahtumia on rajallinen määrä. Tällaisia vaikuttavia tapahtumia voivat olla esimerkiksi jalkapallokohteessa pelaavan joukkueen avainpelaajan loukkaantuminen. Vinttikoirakilpailuissa todennäköisyydet vaihtelevat nopeasti muutaman minuutin ajan ennen lähtöä, vedonlyöjien arvioidessa koirien juoksukuntoa niiden kävellessä kilparadalle esittelyyn ja siitä lähtökoppeihin. Vinttikoirakilpailut vedonlyöntikohteena poikkeavat siis tavanomaisista pre-game vedonlyöntikohteista vedonlyöntipörsseissä siten, että niitä pelataan tyypillisesti vain muutaman minuutin ajan ennen kilpailun lähtöä. Tämän takia vinttikoirakilpailuissa hinnoittelualgoritmin ja koko siihen liittyvän järjestelmän tulee toimia erittäin nopeasti. Vedonlyöntikäyttäytymisellä vinttikoirakilpailuissa on yhtäläisyyksiä in-play kohteiden vedonlyöntiin. Opinnäytetyöstä saatua tietoa yritys voi käyttää tilanteissa, joissa

vetotapahtumia tulee järjestelmään paljon lyhyen ajan sisällä ja kohteen kertoimet muuttuvat nopeasti. Myös in-play vedonlyöntikohteissa joissa lyödään vetoa käynnissä olevasta kilpailussa, muuttuvat kohteen todennäköisyydet nopeasti ottelutapahtumien mukaisesti ja vetoa lyödään paljon lyhyen ajan sisällä. Tällaisessa nopeasti muuttuvassa ympäristössä, tulee kertoimia laskevan järjestelmän vastata nopeasti pyyntöihin ja pysyä toimintakykyisenä suurenkin kuorman alla.

Opinnäytetyö rakentuu seuraavasti. Ensiksi kuvataan vedonlyöntiä, sitten vedonlyöntialan nykytilaa ja toimijoita sekä miten tilaajayritys sijoittuu alalle. Tämän jälkeen kuvataan yrityksen nykyjärjestelmää, siihen opinnäytetyössä tehtäviä laajennuksia sekä kuinka ne toteutettiin. Toteutusten jälkeen järjestelmän toimintaa mitattiin ja analysoidaan järjestelmän toimintaa. Analyysissa havaittuja ongelmakohtia korjataan ja mitataan toimintaa uudelleen. Lopuksi esitellään korjausten vaikutukset järjestelmän toimintaan.

2 Käsitteet ja termit

API	Application programming interface, ohjelmallinen rajapinta, jonka kautta tietokoneohjelmat voivat viestiä toisilleen.
arbitraasi	Tilanne, jossa vedonlyöntikertoimet on asetettu niin että lyömällä kaikkia vaihtoehtoja voittaa varmasti.
back-veto	Pörssissä pelattava veto tapahtuman puolesta.
HTTP	Tiedonsiirtoprotokolla jota käytetään tietoliikenneverkoissa.
in-play	Vedonlyöntikohde, jossa lyödään vetoa käynnissä olevasta tapahtumasta.
kiinteäkertoiminen-peli	Vedonlyönnin muoto, jossa pelaaja saa lyödä vetoa kertoimella, joka ei enää lyöntihetken jälkeen muutu.
lay-veto	Pörssissä pelattava veto tapahtumaa vastaan.
nälkiintyminen	Tilanne jossa rinnakkaisella ohjelmoinnilla tehdyssä järjestelmässä prosessin tai säikeen eteneminen on estynyt vaikka järjestelmä toimii osittain oikein.
Postgre	Relaatiotietokanta
pre-game	Vedonlyöntikohde, jossa lyödään vetoa myöhemmin alkavasta tapahtumasta
Redis	Muistitietokanta
toto-peli	Toto-muotoinen vedonlyönti jossa pelatut rahat jaetaan vedonlyöjille pelattujen määrien mukaisesti.
vedonlyöntipörssi	Sivusto, jossa kuka tahansa voi ostaa tai myydä kohteeseen vetoja valitsemillaan kertoimilla.
yhteyksien varastointi	Tekniikka jossa uudelleen käytetään olemassa olevia yhteyksiä ja vältetään uusien yhteyksien luonti.

3 Vedonlyönti

3.1 Tiedettä vai onnenkauppaa

Vedonlyönti on kiinnostanut ihmisiä jo antiikin aikana. Tällöin ei todennäköisyyksistä vielä ymmärretty mitään vaan voiton tai tappion uskottiin olevan jumalien käsissä ja sattuman varassa. Vedonlyönti oli ihmisille paitsi viihdettä, myös totisinta totta, sillä antiikin Roomassa häviäjä saattoi pelata pahimmillaan omaisuutensa lisäksi myös vapautensa ja päätyä orjaksi. Todennäköisyyksien ymmärtämisen alkuun päästiin vasta renessanssin aikaan 1500-luvulla. (Pinnacle 2019). Tapahtumien todennäköisyyksien ymmärtämisessä vedonlyönnillä onkin ollut merkittävä osa.

Renessanssin aikaan 1500-luvulla eli muuan Gerolamo Cardano, joka oli innokas uhkapelaaja ja vedonlyöjä. Vedonlyönnissä pärjätäkseen hän alkoi mitata sattumanvaraisten tapahtumien todennäköisyyksiä. Cardano oli ensimmäisiä, jotka analysoivat vedonlyöntiä matemaattisin keinoin. Hänen jälkeensä Antoine Gombaud tutki 1600-luvulla nopan silmälukujen todennäköisyyksiä. Todennäköisyyksiä tutkiessaan hän päätyi pyytämään apua ystävältään matemaatikko Blaise Pascalilta. Pascal ja toinen matemaatikko Pierre de Fermat jatkoivat Cardanon työtä ja määrittelivät todennäköisyyksien peruslait. Sveitsiläinen matemaatikko Daniel Bernoulli taas pohti 1700-luvulla miksi ihmiset mieluummin löivät matalariskisempiä vetoja kuin tuottoisempia, eikö vedonlyönnissä tärkeää olisi tehdä voittoa? Pohtiessaan tätä hänelle selvisi, että raha oli eri arvoista eri ihmisille, köyhälle yksi kolikkokin oli vedonlyönnissä merkitsevä toisin kuin rikkaalle. Hän ymmärsi, että panoksen arvo vedonlyöjälle on merkitsevä tekijä panostusta tehdessä ja vetoa lyödessä. Vedonlyönti on ollut mukana kehittämässä monenlaista tieteellistä ajattelua todennäköisyyslaskennasta peliteoriaan ja tekoälyyn. (Kucharski 2016, xi – xiii)

Vedonlyöntiä voi nykyään harjoittaa netin välityksellä sadoilla eri toimijoilla ja vetoa voi löydä tuhansista eri asioista. Yleisimpiä vedonlyöntikohteita ovat urheilutapahtumat. Vetoa lyötäessä vedonvälittäjä antaa kertoimen, jolla maksettu panos maksetaan takaisin. Esimerkiksi lyötäessä vetoa jääkiekko-ottelusta Suomi-Ruotsi vedonvälittäjä voisi tarjota kertoimen 1,9 Suomen voittaessa. Mikäli tätä pelaisi yhdellä eurolla, saisi Suomen voittaessa 1,9 euroa ja Ruotsin voittaessa häviäisi euronsa.

Vedonvälittäjän tavoitteena vedonlyönnissä on tietysti saada tuottoa, kävi vedonlyöntikohteessa kuinka hyvänsä. Tähän vedonvälittäjä pyrkii ensisijaisesti määrittämällä vedonlyöntikohteeseen sopivat kertoimet. Vedonvälittäjä määrittää kohteen kertoimet laskemalla eri

tapahtumien vaihtoehtoille todennäköisyydet. Saadakseen voittoa (ja vähentääkseen taloudellista riskiään), vedonvälittäjä asettaa todennäköisyyksiin määrittämänsä marginaalin. Kohteen kerroin on tämän luvun käänteisluku. Esimerkiksi kolikonheitossa klaavan ja kruunan todennäköisyyksien ollessa molempien 0,5 olisi molempien todellinen (ns. reilu) vedonlyöntikerroin 1/0,5 eli 2. Vedonvälittäjän halutessa tuottoa, voisi vedonvälittäjä asettaa marginaalinsa niin että palautusprosentti olisi 90%. Tällöin molemmille tapahtumille tarjottaisiin kerrointa 1,80. Kerroin (1,8) saadaan ottamalla käänteisluku kertolaskusta todennäköisyys (0.5) * palautusprosentti (90%). Tarjoamalla kerrointa 1,8 molemmille vaihtoehdoille (kruuna ja klaava), jää pelaajien pelatessa vedonvälittäjä voitolle.

Urheiluvedonlyönti on pohjimmiltaan kaupankäyntiä. Osapuolina kaupassa on vedonvälittäjä eli myyjä sekä vedonlyöjä eli ostaja. Kaupankäynti perustuu todennäköisyyksiin, jotka määrittävät tuotteen eli lyötävän vedon hinnan. Vedonvälittäjä kantaa riskiä vaihdon jakautumisesta epätasaisesti kohteen vaihtoehdoille. Jos kerroinvälittäjä tarjoaisi vedonlyöntiä kolikonheitosta, voisi vedonvälittäjä aiemman esimerkin mukaisesti melko huoletta tietää saavansa marginaalin mukaisen tuoton kolikonheiton jakautuessa tasaisesti kruunaksi ja klaavaksi pitkällä aikavälillä. Vedonvälittäjän kannalta valitettavasti näin selviä todennäköisyyksiä ei todellisissa tilanteissa ole. Urheiluvedonlyönnissä vedonvälittäjän kerroinlaskija ei tiedä todennäköisyyksiensä oikeellisuutta. Vedonvälittäjällä ei ole myöskään loputtomasti varaa suuriin tappioihin. Taloudellista riskiä jakaakseen, pyrkii vedonvälittäjä saamaan pelivaihtoa sopivassa suhteessa pelikohteen kaikkiin vaihtoehtoihin. Vedonvälittäjän pyrkimys tasaisesta tuotonjaosta ohjaa kertoimien asettelua, joten urheiluvedonlyönnissä kertoimet eivät kuvasta pelkästään tapahtumien todennäköisyyttä. Kohteen kertoimet kuvastavat myös sitä, miten ihmiset ovat kohdetta pelanneet. (Vuoksenmaa 2016, 64-68)

Vedonvälittäjällä on erittäin suuri määrä kohteita joista lyödä vetoa. Kaikissa kohteissa voi olla vedonvälittäjälle suuri taloudellinen riski. Riskiä hallitakseen kertoimia tulee jatkuvasti muokata. Tämä on vedonvälittäjille työlästä. Las Vegasissa kasinoilla vedonlyöntiä järjestävä Cantor-niminen yritys automatisoi kerroinasettelua Midaaksi nimeämällään algoritmilla. Vedonlyöntikohteiden tilanteiden muuttuessa, muuttuvat Midaan hallinnoimat kertoimet automaattisesti. Midas mallintaa miten ottelun tapahtumat vaikuttavat kohteeseen ja päivittää kohteen kertoimia saamiensa tapahtumien mukaisesti. Cantorin Midas ei pyri ensisijaisesti ennustamaan, miten kohde tulee päättymään, vaan he pyrkivät varautumaan siihen, miten vedonlyöjät pelaavat kohdetta. Tuottoa tehdäkseen Cantorin on tiedettävä mihin vedonlyöjien raha menee ja miten heidän tulisi reagoida kohteessa tapahtuviin muutoksiin. Tietoa kohteista liikkuu pelaajien ja välittäjien välillä ja tehokkaimmat vedonlyönnin tekniikat pidetään visusti salassa. Aasian valtavan suurilla vedonlyöntimarkkinoilla jossa

suurimmat jalkapallovedot lyödään, tieto ja kertoimet poukkoilevat, kun vedonlyöjät ja välittäjät yrittävät selvittää mitä toinen on tekemässä. Alan sisäpiiriläinen kuvaakin Kucharskin kirjassa vedonlyöntialan puskaradiota valtavaksi ja vainoharhaiseksi. (Kucharski 2016, 89-90).

Vaikka vetoa lyödessä lähtökohtaisesti etu on marginaalien vuoksi myyjällä, on ostajalla, eli vedonlyöjällä valinnanvapaus ostaa tai olla ostamatta tarjottua vetoa. Vedonlyöjän etu onkin siinä, että hän voi valita miltä vedonvälittäjällä ja millä kertoimella hän vetonsa ostaa. Kaikkien tarjottavien kohteiden tapahtumien seuraaminen ja kertoimien päivittäminen ovat vedonvälittäjälle hankalaa ja työlästä. Joskus eri vedonvälittäjät tarjoavat samoista vedonlyöntikohteista niin erilaisia kertoimia, että vedonlyöjä voi lyödä (eri vedonvälittäjille) kaikkia kohteen vaihtoehtoja ja jäädä silti voitolle. Nämä ns. Arbitraasi-tilanteet ovat kuitenkin lyhytkestoisia ja kovasti kilpailtuja. Muitakin keinoja lyödä vetoa voitollisesti vedonvälittäjiä vastaan on olemassa. Tilastotieteilijät ovat pitkään tutkineet, miten hyötyä taloudellisesti pelimarkkinoiden tehottomuudesta eli kuinka pelimarkkinoilla voisi tehdä voittoa. (Williams 2004, 63).

Pelimarkkinoiden poikkeavuudet ja niiden aiheuttama tehottomuus ovat tärkeitä keinoja menestyä vedonlyöjänä. Suosikki-altavastaaja-harha, ”The Favorite Longshot” kuvaa tilannetta, jossa vedonlyöjä kuvittelee saavansa paremman ylikertoimen lyödessään vetoa altavastaajan puolesta. Idean kerrotaan saaneen alkunsa vuonna 1949 Richard M. Griffithin analysoidessa tuhansien amerikkalaisten laukkakisojen kertoimia. Tutkimuksissaan hän havaitsi, että pelatessa pienemmillä kertoimilla (”suosikkeja”) olisi pitkällä aikavälillä jäänyt voitolle, tai ainakin hävinnyt vähemmän. Tutkimustulos herätti paitsi vedonlyöjät, myös taloustieteilijät tutkimaan asiaa. Harhan olemassaolo eri pelimarkkinoilla on vahvistettu monissa myöhemmissä tutkimuksissa. (Williams 2004, 63-69).

Uhkapelaajan harhalla tarkoitetaan tilannetta, jossa pelaaja uskoo jo tapahtuneen asian tapahtuvan tulevaisuudessa harvemmin (koska se on jo tapahtunut). Tätä kuvastaisi kolikonheitossa tullut kruuna, jonka jälkeen pelaaja uskoisi klaavan tulevan seuraavaksi hieman todennäköisemmin. Harhaa tutkivat 1996 vinttikoirakilpailuissa Dek Terrell ja Amy Farmer, jotka havaitsivat saavutettavissa olleen 9% tuoton lyömällä lähtökoppia, josta edellisen kisan voittaja oli lähtenyt. Tämä perustui ns. uhkapelaajan harhaan, jossa muut vedonlyöjät välttivät pelaamasta koiraa, jolla oli sama lähtökoppi kuin edellisen lähdön voittajalla. He toimivat näin todennäköisyyksiä vastaan uskoessaan alitajuisesti, että lähdön voittajan lähtökopin täytyy vaihtua. (Williams 2004, 81-84).

Dek Terrell ja Amy Farmer ovat tutkineet myös lähtökopin vaikutuksesta koirakilpailun voittajaan, voisiko tiettyä lähtökoppia järjestelmällisesti lyömällä jäädä voitolle? Ajatus kuulostaa järjettömältä, eikö tällaisen harhan paljastuminen olisi nopeasti hyväksikäytettävissä? Heidän tutkimustensa tulokset osoittivat, että voittorahat eivät jakaudu tasaisesti kaikille lähtökopeille. Sisäratojen lähtökopit antoivat merkittävästi paremman voittotuoton kuin ulkoratojen huonoimmat lähtökopit. Yksi syy tähän heidän mukaan on, että vedonlyöjät ajattelevat parhaiden koirien olevan ulkoradoilla ja pelaavan näitä merkittävästi liikaa jolloin sisemmille lähtökopeille rahaa pelattiin todennäköisyyksiin nähden liian vähän. (Williams 2004, 91-92).

Esitellyn kaltaisia harhoja ja poikkeavuuksia esiintyy vedonlyöntimarkkinoilla aiheuttaen markkinoiden tehottomuutta. Vedonlyöntimarkkinoiden tehokkuudella tarkoitetaan sitä kuinka hyvin kohteen kerroin vastaa sen todellista todennäköisyyttä. Markkinoiden tehokkuus on riippuvainen myös kohteen suosiosta vedonlyöjien keskuudessa. Vähemmän suositut Amerikan baseball sarjan MLB:n runkosarjaottelut voivat olla merkittävästi tehokkaammat kuin amerikkalaisen jalkapallon Super Bowl-tapahtuman kertoimet. Tämä voi johtua siitä, että Super Bowl tapahtumassa harrastevedonlyöjien määrä suhteessa fiksuihin on huomattavan paljon suurempi kuin MLB-ottelussa. King Yaon mukaan pelimarkkinoiden tehokkuuteen vaikuttaa kaksi asiaa: fiksun rahan määrän suhde harrasterahaan ja kohteeseen liittyvän tiedon määrän ero kerroinlaskijoiden ja ammattipelureiden välillä (Yao 2008, 62)

Menestyvillä ammattipelaajilla on vaikeuksia löytää vedonvälittäjää, jotka hyväksyvät heidän vetoja. Tai vaikka vedonvälittäjä ottaisikin vastaan ammattivedonlyöjältä vedon, on pelaajan panoskatto rajoitettu niin matalaksi, ettei se ammattipelaajalle ole riittävä. Pinnacle Sports on kuitenkin alusta alkaen hyväksynyt ja jopa rohkaissut ammattipelaajia asiakkaikseen. Vaikka pelaaja olisikin toistuvasti voittanut, ei Pinnacle sulje pelaajan tiliä tai rajoita tämän maksimipanosta. Yleisesti vedonvälittäjät katsovat kohteessa tapahtuvaa vedonlyöntiä, mutta Pinnacle käyttää paljon vaivaa saadakseen paremman ymmärryksen siitä kuka vedon on lyönyt. Hyväksyessään ammattipelaajat asiakkaikseen, ymmärtää vedonvälittäjä mitä nämä pelaajat ajattelevat kohteen todennäköisyyksistä. He tietävät, että joskus asiakkaat tietävät vedonvälittäjää paremmin mitä pelissä tulee tapahtumaan. Pinnacle tietää kohteet avatessaan, etteivät kohteen kertoimet ole täydelliset. Saadessaan fiksuilta pelaajilta informaatiota, he voivat säätää omia todennäköisyysarvioitaan paremmiksi. Pinnacle saa hyvän varmuuden omiin kertoimiinsa yhdistämällä omat tilastolliset ennusteensa ja fiksuilta pelaajilta keräämänsä tiedon, ja voi näin ollen ottaa vastaan isojakin vetoja. Voikin sanoa, että Pinnacle maksaa fiksuille pelaajille heidän antamasta tiedosta. (Kucharski 2016, 91-93).

Vaikka vedonlyönti alana on yleisesti paheksuttua, kiinnostaa se monia tutkijoita ja tieteentekijöitä vielä nykyäänkin. Historiallisesti vedonlyönti on kehittänyt matematiikkaa, tilastotiedettä ja todennäköisyysteoriaa. Nykyään tutkijat käyttävät pokeria tutkiakseen tekoälyä, kaupankäyntialgoritmit tekevät päätöksiä vedonlyönnistä ja urheilusta kerätään dataa, joilla analysoidaan joukkueiden yksittäisten pelaajien toiminataa. Tiede ja vedonlyönti jatkavat kehittymistään yhdessä. (Kucharski 2016, 217).

3.2 Vedonlyöntiala numeroina

Vedonlyönti on hyvin suosittua ympäri maailman. Vetoa voi lyödä hyvin monenlaisista kohteista, mutta urheiluvedonlyönti on vedonlyönneistä yleisin. Euroopan säänneltyjen pelimarkkinoiden kooksi on arvioitu vuonna 2013 noin 90 miljardia euroa, josta noin 20% on vedonlyöntiä. Sääntelemättömät markkinat ovat kuitenkin vielä merkittävästi suuremmat, yksistään Aasian sääntelemättömien markkinoiden kooksi arvellaan 500 miljardia USD. (Egba 2013.). Iso-Britanniassa, jossa vedonlyöntipörssit ovat hyvin suosittuja, on työntekijöitä pelialalla 106 366. Alan tuotto Iso-Britanniassa oli huhtikuusta 2017 – maaliskuuhun 2018 14,4 miljardia GBP. (Gambling Commission 2018.)

Suomessa vedonlyöntiä toimeenpanee Veikkaus jonka tulos 2017 oli 1021,3 miljoonaa euroa. Kiinteäkertoimisia kohteita (esim. Pitkäveto) oli pelattu 379,9 miljoonaa euroa. Suomessa pelimarkkinat on asukaslukuun suhteutettuna suuri, tästä tosin suuri osa myydään lottona (344,8 milj. Euroa) ja Euro jackpotissa (213,9 milj. Euroa). (Veikkaus 2017.)

3.3 Miten asiakasyritys toimii alalla

Asiakasyritys kehittää vedonlyöntialalle IT-palvelua, joka laskee todennäköisyyksiä ja kertoimia vedonlyöntikohteisiin. Asiakasyritys tarjoaa palveluaan peliyhtiöille, jotka voivat käyttää kertoimia omien vedonlyöntikohteiden hinnoitteluun ja riskinhallintaan. Yrityksen taustalla ovat kertoimien laskentaa varten kehitetyt algoritmit, jotka optimoivat tulosta ja hallinnoivat riskiä. Algoritmit käyttävät hyväkseen tietoja vedonlyöntikohteesta: kohteessa tapahtuvia myyntejä, markkinahinnoissa tapahtuvia muutoksia sekä muita tietoja joita kohteesta on tarjolla. Vedonvälitysyrityksistä Euroopassa moni hoitaa riskinhallinnan rajoittamalla voittavia pelaajia. Yritys uskoo, että voittavien pelaajien tietoa tulee käyttää kerroinasettelussa hyväksi, kuten Pinnacle Sports onnistuneesti tekee yhdistäessään omat tilastolliset mallinsa pelaajilta saatuun tietoon. Yrityksen palvelu kasvattaa peliyhtiön tuottoa mukauttamalla kohteen kertoimia optimaalisesti ja automaattisesti.

3.4 Vedonlyöntipörssit

Vedonvälittäjät ovat saaneet vedonlyöntipörsseistä 2000-luvulla kilpailijan toiminnalleen. Vedonlyöntipörssit ovat lähinnä Brittien saarilla toimivia verkkosivustoja (-palveluita), joihin kuka tahansa voi jättää vetotarjouksen tai pelata tarjolla olevaa vetokohdetta. Vedonlyöntipörssit toimivat hieman kuin pörssit, osakkeiden sijaan kauppaan käydään vedonlyöntikohteiden hinnalla. Tunnetuin pörsseistä on Betfair, joka Kucharskin mukaan käsittelee yli seitsemän miljoonaa vetoa päivässä. (Kucharski 2016,93)

Vedonlyöntipörssin sivulla on näkyvillä kohteessa tarjolla olevat kertoimet ja maksimipanokset joilla niitä voi pelata. Mikäli ostat tarjolla olevan kertoimen asian tapahtumisen puolesta, voit valita haluamasi kohteen tarjolla olevan kertoimen ja haluamasi summan. Tapahtuman puolesta lyömistä kutsutaan pörssissä nimellä ”Back”-veto.

Pörssissä voit pelata myös asian tapahtumista vastaan. Jos haluat jättää kohteeseen tarjolle, että asia ei tapahdu, voit valita haluamasi kertoimen sekä maksimitappiosi. Nämä määrittävät kertoimen ja panoksen, joilla joku muu voi pelata sinua vastaan. Tapahtumaa vastaan lyömistä kutsutaan pörssissä nimellä ”lay”-veto.

Koska voit pelata asian tapahtuman puolesta ja vastaan, on mahdollista jäädä pelikohteessa voitolle jo ennen tapahtuman alkua. Jos joukkueelle A tarjotaan pörssissä kerrointa 5 ja pelaat sitä 10 eurolla, saat 50 euroa takaisin, jos joukkue A voittaa. Jos ennen ottelua tapahtuukin jokin merkittävä muutos, esim. joukkueen B tähtipelaaja loukkaantuu ja A:n kerroin muuttuu ja on nyt 2. Voit nyt pelata A:n voittoa vastaan hyväksymällä vetotarjouksen 10 euroa kertoimella 2, ja päädyt tilanteeseen jossa A:n voittaessa voitat ensimmäisestä vedosta 50 euroa ja häviät toisesta 20 euroa. Joukkueen B voittaessa häviät ensimmäisestä vedosta 10 euroa ja voitat toisesta vedosta 10 euroa. (Kucharski 2016, 95)

Vetotarjouksen jättäjä tai pelaaja ei koskaan tiedä ketä vastaan on pelannut, vaan vedonlyöntipörssi yhdistää tarjouksen ja pelaamisen, sekä hoitaa rahansiirron pelaajien pelitilien välillä kohteen päättyessä. Vedonlyöntipörssit perivät välittämistään vedoista komission, joka vaihtelee pörsseittäin. Komissio vaihtelee pörssistä riippuen noin 1-5% voitetusta tai pelatusta summasta.

Vedonlyöntipörssissä voi pelata verkkosivujen kautta, jolloin näet sen kohteen sen hetkiset hinnat. Toinen vaihtoehto on kytkeä tietokoneohjelma suoraan vedonlyöntipörssiin, jolloin ohjelma voi asettaa kohteeseen vetoja automaattisesti. Tätä varten vedonlyöntipörssit

tarjoavat APIn, jonka kautta tietokoneohjelmat voivat viestiä ohjelmallisesti vedonlyöntipörssin kanssa. Tämänkaltaiset vedonlyöntirobotit ovat nopeampia ja voivat pelata kymmeniä pelejä yhtä aikaa. Robotit pelaavat virheelliset hinnat niin nopeasti pois, että verkkosivuston kautta ei väärin asetettuja hintoja ehdi todennäköisesti edes nähdä. Robotitkin voivat kuitenkin tehdä virheitä, ja virheet voivat olla potentiaalisesti isojakin. (Kucharski 2016, 113)

Vedonlyöntipörsseissä on tarjolla samoja kohteita kuin suosituissa perinteisillä vedonlyöntisivustoillakin. Pelatuimpia kohteita ovat jalkapallon suuret kilpailut ja suositut ottelut (mestareidenliiga, valioliiga). Euroopassa toimivia vedonlyöntipörssejä ovat Betfair, Matchbook, Betdaq ja Smarkets. Suurin ja suosituin pörsseistä on Betfair ja muut ovat selkeitä haastajia. Vedonlyöntipörsseissä käydään kauppaa vuosittain miljardeilla euroilla. Betfairin haastajista Smarkets on julkisesti kertonut, että vuonna 2016 heidän pörssissä on tehty kauppaa 2,6 miljardilla GBP:llä. (Business Insider 2016.)

3.5 Vinttikoirakilpailut

Vinttikoirakilpailussa koirat juoksevat ovaalin muotoisella suljetulla radalla sähköisen ”jäniksen” perässä. Lajin kehitteli O.P. Smith Kaliforniassa 1919, ja se esiteltiin Englannissa 1926. Laji keräsi suosiota Englannissa ja tulikin suosituimmaksi kuin Amerikassa. Kilpailussa on tavallisesti 8 lähtöä ja lähtöjen pituudet vaihtelevat 210 metristä 1100 metriin. Kilpailuja juostaan tasaisella radalla, sekä radalla jossa on esteet joiden yli koirat hyppäävät. Yleensä lähdössä on juoksemassa enintään kuusi koiraa. (Britannica 2018)

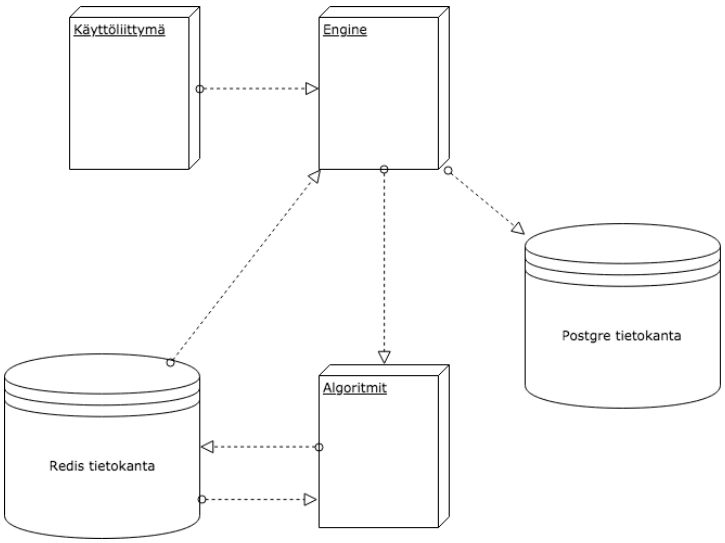
Vinttikoirakilpailut ovat Iso-Britanniassa suosittu vedonlyöntitapahtuma. Lähtöjä järjestetään päivittäin kymmeniä useassa eri kaupungissa. Vetoa voi lyödä niin verkossa kuin paikan päällä toto-pelinä sekä kiinteäkertoimisena pelinä. Vedonlyöntipörssissä ja tässä työssä käytetty pelimuoto on kiinteäkertoiminen pre-play.

Kehitysympäristönä tässä työssä käytetyssä vedonlyöntipörssissä pelivaihto vinttikoirakohteissa vaihtelee tuhansista euroista jopa kymmeniin tuhansiin euroihin per lähtö. Vinttikoirakohteita löytyy kaikista Euroopan alueella toimivista vedonlyöntipörsseistä.

4 Yrityksen järjestelmän kuvaus

Yrityksen järjestelmä laskee erilaisille vedonlyöntikohteille syötteenä tulevien tietojen perusteella kohteen todennäköisyyksiä ja tilanteeseen sopivia kertoimia. Arkkitehtuurillisesti järjestelmä rakentuu matemaattisten kerroinpäivitysalgoritmien ympärille.

Järjestelmän voi jakaa neljään loogiseen osaan: käyttöliittymään, moottoriin, algoritmeihin sekä tietovarastoon. Järjestelmän osat sekä niiden välinen viestintä on kuvattu kuvassa Nykyjärjestelmän arkkitehtuuri.



Kuva 1 Nykyjärjestelmän arkkitehtuuri

Järjestelmässä on WWW-pohjainen käyttöliittymä, jolla syötetään järjestelmään uusia tietoja ja kontrolloidaan järjestelmän toimintaa. Käyttöliittymässä on näkymä järjestelmän tämänhetkisestä tilasta ja käynnissä olevista kohteista. Käyttöliittymä lähettää tietoja ja käskyjä moottorille, joka prosessoi eri integraatioista tulevia tietoja ja syöttää niitä eteenpäin. Lisäksi moottori huolehtii tehtävien ajamisesta ja niiden rinnakkaistamisesta. Moottorin tehtävä on myös huolehtia ajastetusta tietojen hakemisesta eri kohteista. Moottoriin on rakennettu useita integraatioita ulkoisiin palveluihin, joiden kautta järjestelmään saadaan uutta pelikohteisiin ja niiden kertoimiin liittyvää tietoa. Algoritmeja ajetaan moottorin lähettämien käskyjen mukaisesti. Algoritmit ja moottori käyttävät välimuistina Redis-tietokantaa, joka tunnetaan nopeana key-value tietovarastona. Tiedon pidempiaikaiseen säilytykseen käytetään Postgre SQL-tietokantaa. Postgre-tietokanta on järjestelmän tiedon loppusäilytyspaikka, jonne tietoa siirretään Redis-kannasta säännöllisesti.

5 Laajennoksen suunnittelu

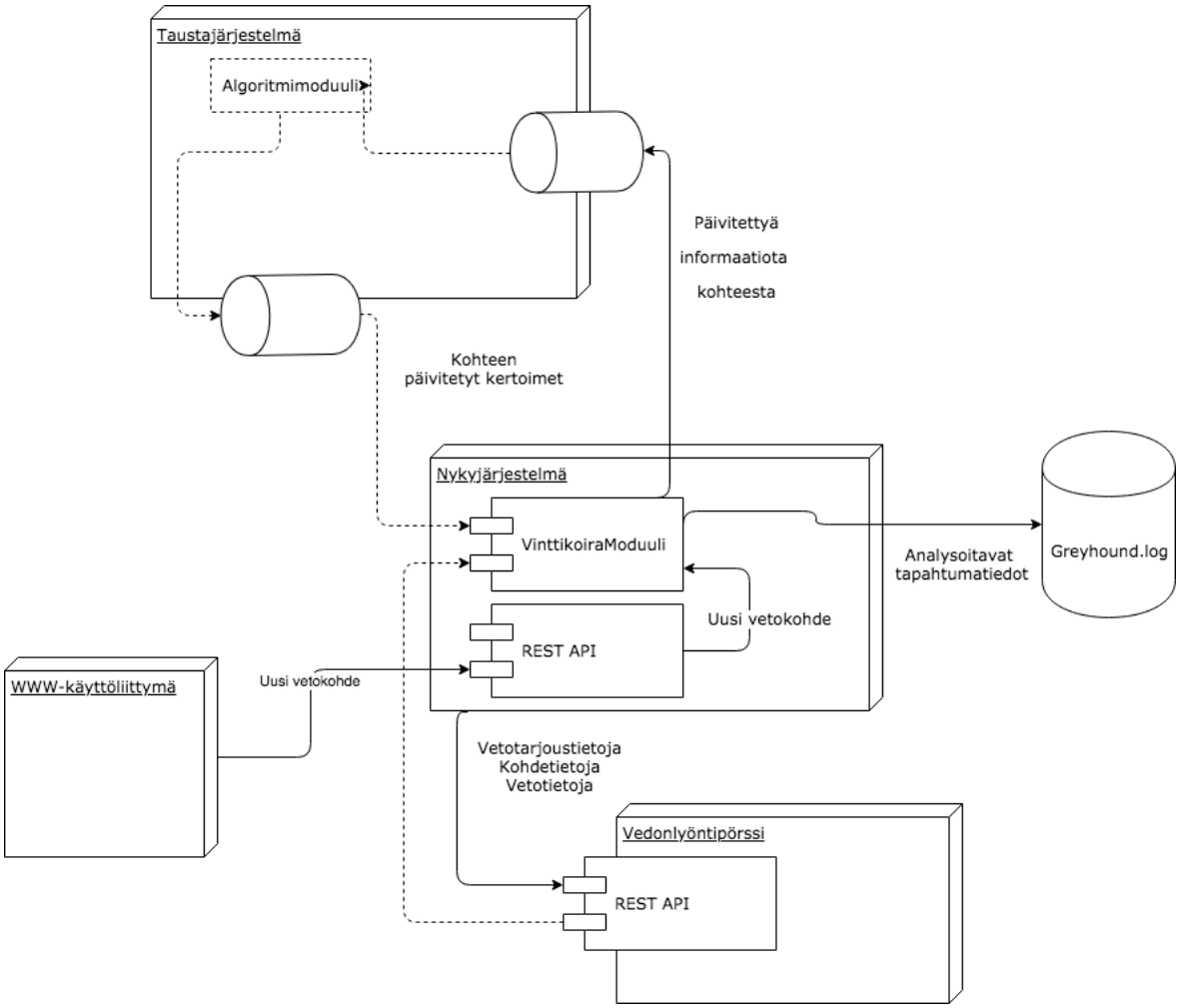
5.1 Vaatimukset

Yritys hakee toiminnalleen kasvua vedonlyöntipörsseistä, ja nykyjärjestelmän arkkitehtuuri ja toiminta on suunniteltu alun perin eri tarpeisiin. Kasvua pörsseistä haetaan monista kohteista. Yksi mielenkiintoisimmista kohteista on vinttikoirakilpailukohteet. Vinttikoirakilpailut ovat mielenkiintoisia, koska kohteet vaikuttavat olevan hieman tehottomat, niitä järjestetään päivittäin ja kohteita on paljon. Ennen toiminnan laajaa aloitusta, tulee palvelua ja algoritmeja merkittävästi kehittää. Toisaalta on myös epävarmuutta, onko nykyinen tekninen toteutus ja arkkitehtuuri riittävän nopea tällaista toimintaa varten. Opinnäytetyössä tehtävän laajennoksen avulla yritys hakee tietoa siitä mitä vinttikoirakohteissa on mahdollista tehdä nykyisellä arkkitehtuurilla ja millaisia mahdollisia muutoksia laajempi toiminta vinttikoirakohteissa vaatisi yrityksen järjestelmään ja arkkitehtuuriin. Koska vinttikoirakilpailut ovat poikkeuksellisen nopeatahtisia vedonlyöntitapahtumia vedonlyöntipörsseissä, asettavat ne järjestelmälle erilaisia haasteita suoritusaikojen ja rinnakkaisuuden osalta kuin muut tähän asti kokeillut vedonlyöntikohteet. Varsinainen aktiivinen pelaaminen vinttikoirakohteissa kestää vain muutaman minuutin alkaen 2-4 minuuttia ennen kilpailun lähtöä.

Laajennoksen toimintaa varten järjestelmälle syötetään tiedot uudesta vinttikoirakilpailusta web-käyttöliittymän kautta. Kilpailun syötteenä annetaan kohteen ID vedonlyöntipörssissä, ja jokaiselle kohteen kilpailijalle lähtötodennäköisyys. Kun tiedot on syötetty, lähetetään tiedot nykyjärjestelmälle, joka laskee jokaiselle kilpailijalle kertoimet ja palauttaa tiedot optimikertoimesta sekä panoskoosta. Näiden perusteella jätetään vedonlyöntipörssiin vetotarjous. Vedonlyöntipörssin rajapinnasta kysellään toistuvasti, onko jätettyjä vetotarjouksia pelattu. Kun havaitaan, että jätettyä tarjousta on pelattu, lähetetään pelatun vetotarjouksen kerroin ja panostiedot nykyjärjestelmälle. Nykyjärjestelmän laskee uudet kertoimet ja panoskoot, joiden perusteella jätetään pörssiin uudet vetotarjoukset. Lisäksi seurataan jatkuvasti kohteen(kilpailun) tilaa vedonlyöntipörssissä ja kilpailun alkaessa (kohteen sulkeutuessa) lopetetaan kohteen kertoimien laskenta ja vetotarjousten jättäminen.

5.2 Rajaus

Laajennoksen tarkoituksena on siis saada testattua järjestelmän toimintaa rajatusti vinttikoirakilpailukohteissa vedonlyöntipörssissä. Tärkeintä antia on saada parempi ymmärrys nykyjärjestelmässä ja sen arkkitehtuurissa olevista rajoituksista sekä aikaviiveistä, joita Internetin välityksellä toimiessa tapahtuu yrityksen järjestelmän ja vedonlyöntivälittäjien välillä. Useita perustoimintoja vedonlyöntipörssissä toimimiseen on nykyjärjestelmässä jo toteutettu, mutta vinttikoirakilpailuja varten tässä työssä laajennetaan nykyjärjestelmää tarpeellisilta osin. Laajennukset tehdään Java-kielellä ja ajetaan osana nykyjärjestelmää.



Kuva 2 Nykyjärjestelmän arkkitehtuuriin sekä laajennoksen sijoittuminen siinä

5.3 Analyysia varten kerättävät tiedot

Kun järjestelmä saadaan jättämään vetotarjouksia ja reagoimaan niihin vinttikoirakilpailuissa, kerätään tietoa järjestelmän tähän liittyvien osasuoritusten kestoajoista. Osasuorituksia ja niiden kestoa seurataan millisekuntitasolla tallettamalla päivitystapahtumassa olevien osatapahtumien toiminnallisuus tapahtumakuvauksineen lokitiedostoon (Arkkitehtuuri 2, Greyhound.log).

Osasuoritus kirjoittaa lokitiedostoon suorituksen alkamisajankohdan ja päättymishetken. Näiden avulla mitataan osasuoritusten suoritusaikoja vinttikoirakilpailujen vedonlyönnin ollessa käynnissä. Osasuoritusten suoritusaikojen analysointi tehdään ohjelmallisesti keräämällä mitatut tiedot erillisestä lokitiedostosta, ryhmittelemällä ne ja laskemalla suoritusten kestoista avainlukuja. Nykyjärjestelmä on kirjoitettu suureksi osin Java-kielellä, ja vinttikoirakilpailuja varten tehdyt laajennukset kirjoitetaan Javalla, sekä käyttöliittymän osalta JavaScriptillä ja HTML-kuvauskielellä.

5.4 Käyttötapaus

Järjestelmässä on valmiina www-käyttöliittymä, jonka kautta voi pienellä laajennoksella syöttää vinttikoirakilpailun perustiedot. Syötetyn vedonlyöntipörssin market Idn avulla järjestelmä löytää vedonlyöntipörssistä oikean vedonlyöntimarkkinan ja hakee kohteeseen liittyviä tietoja, kuten alkamisajankohta, kilpailijoiden nimet ja kilpailun tilan näkyviin käyttöliittymään. Uusi kohde syötetään järjestelmään www-käyttöliittymän avulla.

Taulukko 1 Tärkeimmät pelikohdetta koskevat syötteet

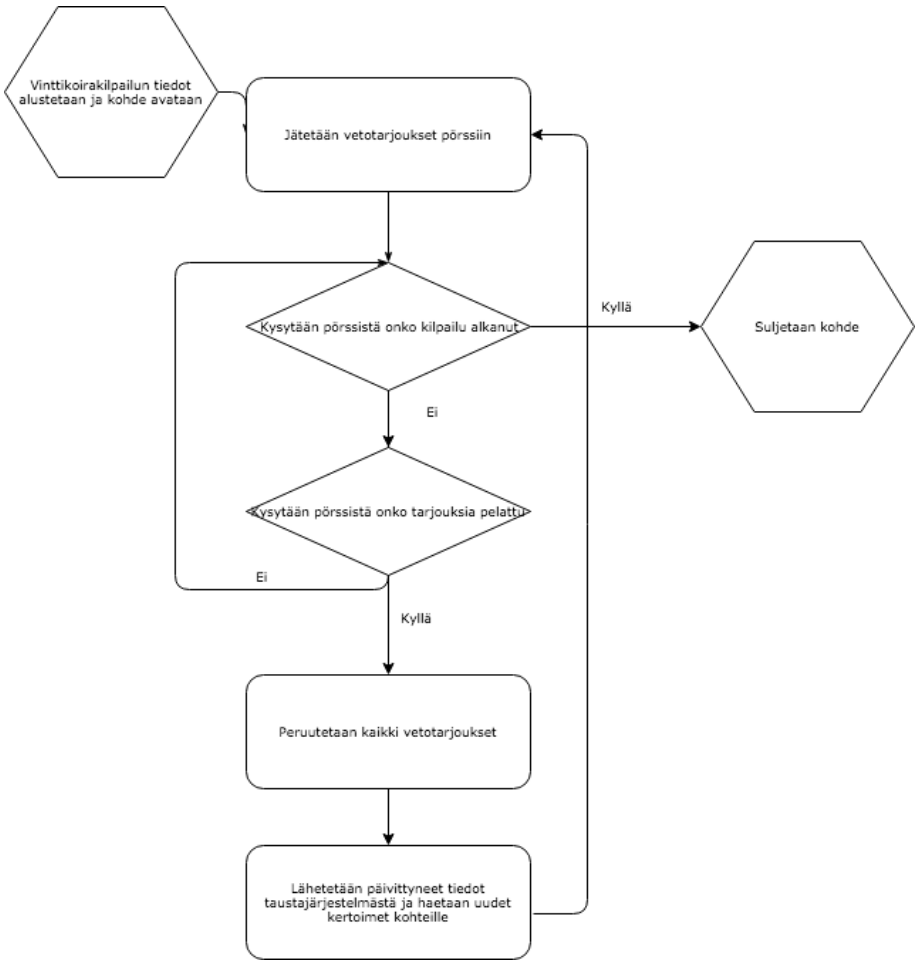
Parametri	Tietotyyppi	Kuvaus
integrationMarketId	String	Kohteen ID vedonlyöntipörssissä
marketStartTime	String	Kohteen alkamisaika
competitionName	String	Kohteen kilpailun nimi
minMarginPerSelection	Double	Käytetty marginaali min-arvo
maxMarginPerSelection	Double	Käytetty marginaali max-arvo

Pelikohteeseen voi liittyä N-kpl kilpailijoita, joiden puolesta tai vastaan voi lyödä vetoa tai jättää vetotarjouksia.

Taulukko 2 Tärkeimmät kilpailijaa koskevat syötteet

Parametri	Tietotyyppi
selectionID	String
selectionName	String
probability	Double

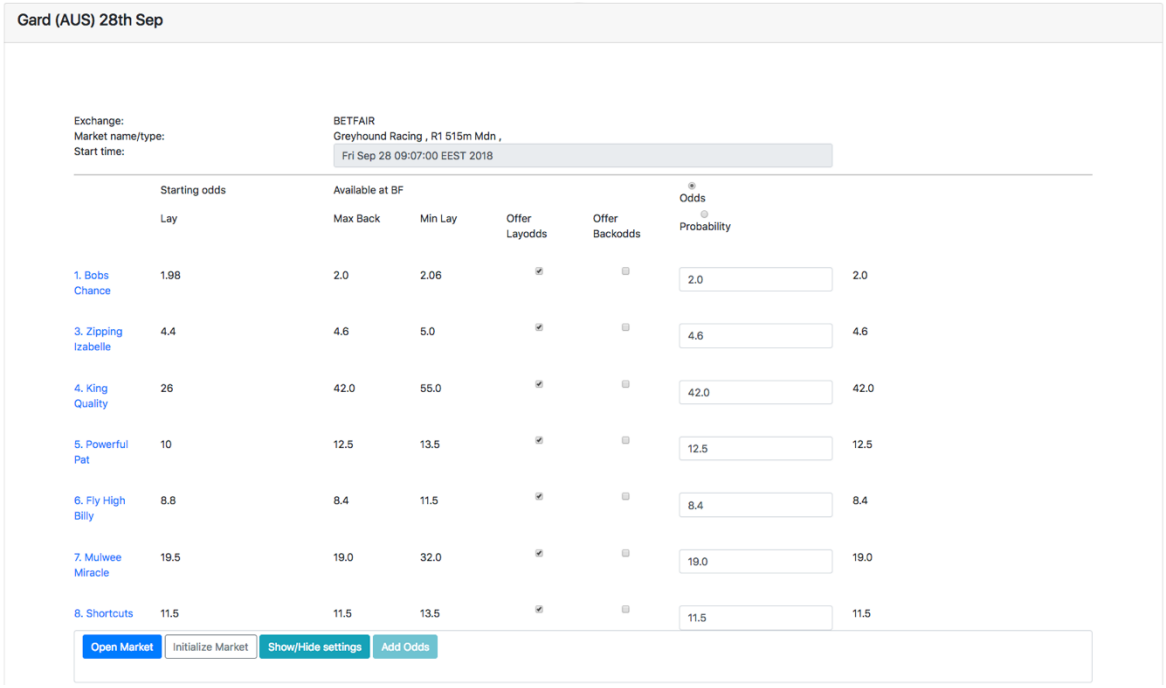
Kun pelikohteen perustiedot on syötetty järjestelmään ja kohde on avattu alkaa järjestelmä jättää vetotarjouksia vedonlyöntipörssiin yrityksen nykyjärjestelmästä saatujen kertoimien avulla. Järjestelmän annettua kohteelle uudet kertoimet, päivitetään uudet vetotarjoukset vedonlyöntipörssiin. Vetoja tarjotaan kaikille kohteille noin 3 minuutin ajan ennen vinttikoirakilpailun lähtöä, jolloin varsinainen aktiivinen pelaaminen tapahtuu. Vinttikoirakilpailujen voittajakohteessa jätetään lay-tyyppisiä vetotarjouksia kaikille voittajavaihtoehdoille. Lay-tyyppisiä vetotarjouksia jättämällä vedonlyöntipörssin muut pelaajat voivat lyödä haluamansa vinttikoiran voiton puolesta ns. back-vedon. Kun jotain vetotarjousta pelataan, järjestelmän tulee peruuttaa kaikki kohteen sillä hetkellä voimassa olevat vetotarjoukset ja toimittaa eteenpäin tiedot pelatusta vetotarjouksesta. Kun järjestelmä on laskenut uudet, päivitetyt kertoimet, jätetään kaikille kohteen kilpailijoille uudet vetotarjoukset vedonlyöntipörssiin. Toiminta on esitetty vuokaaviona kuvassa 3.



Kuva 3 Kuvaus toiminnasta vinttikoirakilpailussa1

6 Järjestelmään tehdyt muutokset ja laajennukset

Järjestelmässä oli valmiina toimintoja, jotka mahdollistavat pelaamisen vedonlyöntipörssissä. Vinttikoirakilpailujen poiketessa jonkin verran muusta vedonlyöntitarjonnasta, jouduin tekemään joitakin muutoksia ja laajennuksia järjestelmään, jotka pyrin kuvaamaan tässä lyhyesti ja selkeästi.



Kuva 4 Kohteen syöttäminen

Kuvassa Kohteen syöttäminen on kuvattu vinttikoirakilpailukohteen syöttämistä järjestelmälle. Järjestelmälle on lähtösyötteenä annettu kohteen vedonlyöntipörssin ID, jonka perusteella vedonlyöntipörssistä on haettu kohteen perustiedot: alkamisaika, kohteen lajin ja kilpailijoiden nimet. Käyttäjä on syöttänyt kilpailijoiden riveillä tekstikenttään lähtötodennäköisyyden kerroinmuodossa ja painanut ”Initialize market”-nappia. Nappi on lähettänyt nykyjärjestelmälle tiedot uudesta kohteesta ja todennäköisyyksistä. Lasketut kertoimet näytetään sarakkeessa ”Starting odds lay”. Käyttäjän painaessa ”Open Market”-nappia, aloitetaan vetotarjousten jättäminen vedonlyöntipörssiin.

Vinttikoirakilpailun vedonlyönnin tapahtuessa lyhyessä ajassa, tein kohteen syöttämisen käynnistystä nopeuttavia muutoksia, esim. Suoraviivaistin käyttöliittymän toimintaa vähentämällä vaadittuja klikkauksia kahdesta yhteen. Tätä varten kirjoitin ohjelmapätkän, joka haki vedonlyöntipörssistä tarjolla olleita tietoja kohteen vaihtoehdoista. Näin pystyin automatisoimaan käyttöliittymässä käyttäjän toisen napin painalluksen (”Initalize market”).

Lisäsin taustajärjestelmään ohjelmakoodia, joka tunnisti vedonlyöntikohteen vinttikoirakilpailuksi ja vaihtoi sille algoritmin vaatimat vinttikoirakilpailuissa käytettävät parametrit. Kehittäessäni havaitsin muutamia pienkehitystä vaativia muutoksia nykyjärjestelmässä, jotka olivat tehtävä, jotta vinttikoirakilpailua pystyi käyttämään järjestelmän kohteena. Kokonaisuutena nykykäyttöliittymä toimi kuitenkin hyvin vinttikoirakilpailukohteisiin, ja vaati olettamaani vähemmän muutoksia.

Järjestelmän moottori vaati pieniä muutoksia ohjelmakoodiin. Järjestelmän moottorissa on osa, joka huomaa onko järjestelmän tarjoamia vetoja pörssissä pelattu ja onko kilpailu jo alkanut. Alkuperäisestä suunnitelmastani poiketen pystyin käyttämään nykyistä versiota pienillä laajennuksilla, eikä uutta moduulia tarvinnut tehdä. Vinttikoirakilpailukohdetta varten nopeutin kuitenkin ohjelmakoodia vetotarjousten huomaamista, jotta pelattuihin vetoihin reagointi olisi nopeaa. Lisäksi uudelleenkirjoitin vinttikoirakilpailua varten kohteen sulkemisen järjestelmässä.

Järjestelmän toiminnan analyysia varten tutkin järjestelmän päivitystoiminallisuuden ohjelmakoodia, ja jaoin sen osasuorituksiin. Käyttäen SLF4j –kirjastoa loin nimeämäni GreyhoundLoggerin, joka kirjoitti osasuoritusten alkamiset ja päättymiset omaan lokitiedostoon. Tämä vaati muutoksia järjestelmän käännös ja paketointi osaan, jotta GreyhoundLoggerin konfigurointimuutokset päivittyivät oikein tuotantopalvelimelle.

7 Mittaukset ja toiminnan analysointi

7.1 Tietojärjestelmän toiminnan analysointi

Tietojärjestelmästä riippumatta hyviä, järjestelmältä haluttuja ominaisuuksia ovat nopea vasteaika, toiminnan nopeus ja skaalautuvuus. Näiden ominaisuuksien hyvä laatu muodostaa kilpailuedun ja erottaa järjestelmän kilpailijoistaan aivan kuin lisätoiminallisuudet. Käyttäjät arvostavat nopeutta ja helppokäyttöisyyttä. Järjestelmän toiminnan tehokkuus eli nopeus voi olla jopa elinehto järjestelmän toiminnalle. Arvopaperikauppaa tekevän järjestelmä tulee pystyä käsittelemään suuri määrä transaktioita sekunnissa ollakseen hyödyllinen. Tällaisessa arvopaperikauppaa tekevässä järjestelmässä hidastelu tarkoittaa menetettyjä kaupankäyntimahdollisuuksia ja hidasteleva järjestelmä voikin viedä liiketoiminnalta kannattavuuden. Järjestelmän tehokkuuden mittaaminen ja analysointi tukevat järjestelmäarkkitehtuurin kehitystä ja vie järjestelmän teknologiakehitystä eteenpäin. Mittaamisella voidaan mm. selvittää järjestelmän toimintaa ruuhkatilanteista, mitä vaikutuksia uusilla toiminallisuuksilla on järjestelmään ja täyttääkö järjestelmä sille asetetut odotukset ja vaatimukset. (Bondi 2015, Chapter 1).

Vedonlyöntipörssissä toimivalle vedonlyöntijärjestelmälle asetetaan samankaltaisia vaatimuksia kuin arvopaperikauppaa tekevälle järjestelmälle. Järjestelmän toiminnan tulee olla tehokasta, nopeaa ja skaalautuvaa. Kannattavuuden vedonlyöntipörssissä mahdollistaa vain tehokkaasti toimiva järjestelmä, ja kuten arvopaperikaupassakin, hidastelu merkitsee menetettyjä kaupankäyntimahdollisuuksia vetojen jäädessä lyömättä.

Tietojärjestelmän toiminnan analysointi aloitetaan tavallisesti perustoiminallisuuden testaamisesta, jolla järjestelmän oikea toiminta varmistetaan. Järjestelmän toimiessa suunnitellusti selvitetään, että järjestelmä toimii riittävän tehokkaasti sopivilla kustannuksilla. Jotta järjestelmän riittävä tehokkuus voidaan todeta, tulee järjestelmän toimintaa mitata ja ennustaa tulevaisuuden tarpeita. Millainen on riittävän tehokas järjestelmä, mikä on sille sopiva kustannus. Ymmärtääksemme paremmin järjestelmän mittauksia ja niiden mittausten tuloksia, tulee mittauksia aloittaessa määrittää konteksti, jossa asioita tutkimme. Kontekstiin kuuluu ympäristö, jossa järjestelmää ajetaan, ympäristön ja järjestelmän komponentit ja mitattavat parametrit jotka meitä kiinnostavat. (Fortier & Michel 2003, 120.)

Järjestelmän tehokkuuden mittaamisessa on tärkeää mitata aitoa järjestelmää, jotta mittauksilla saadaan tilanteesta oikea kuva. Järjestelmää tulee mitatessa käyttää samalla tavalla, kuin sitä oikeassakin käytössä käytettäisiin. Järjestelmän mittaamista voi Oaksin mukaan jakaa kolmeen kategoriaan. Ensimmäinen kategoria on microbenchmarks, jolla mitataan hyvin pienen osan toiminnan tehokkuutta. Microbenchmarks voisi tarkoittaa järjestelmän yhden funktion tehokkuuden mittausta. Tämä saattaa antaa kuitenkin merkittävästi väärän kuvan testiympäristön poiketessa aidosta järjestelmästä. Toinen kategoria on macrobenchmarks, jolla mitataan itse järjestelmän toimintaa ulkoisten osiensa kanssa. Monimutkaisten järjestelmien toiminta voi kuitenkin olla erilaista, jos niiden osia jätetään testistä pois. Tämän takia mitatessa tehokkuutta on hyvä käyttää järjestelmää kokonaisuudessaan. Koko järjestelmän toimintaa testatessa toimintaympäristön resurssien mahdolliset pullonkaulat paljastuvat aidossa käyttötilanteessa. Kahden edellisen kategorian välimalli on mesobenchmarks, jolla testataan järjestelmän osan toiminallisuutta, vaikka järjestelmän kaikki osat eivät olekaan testin aikana käytössä. Esimerkkinä mesobenchmarks-testistä olisi kuinka nopeasti JSP-sivu ladataan palvelimelta silloin kun muut osat eivät ole käytössä. (Oaks 2014, 11-19.)

Kun järjestelmän toimintaa mitataan, on tärkeää valita mittareiksi sellaisia arvoja, jotka ymmärretään yleisesti kuvaamaan järjestelmän toimintaa. Kun valitut arvot ovat selkeitä ja tunnistettuja, voidaan järjestelmän toiminnan tehokkuutta tarkastella monelta eri näkökannalta vertailemalla mitattuja arvoja. Mittareita voidaan tarkastella niin liiketoiminnan, kuin ohjelmistokehittäjän näkökulmasta. Erilaiset tarpeet johtavat siihen, että samoja mittareita

saatetaan haluta tarkastella hyvin eri tavoin. Kehittäjää voi kiinnostaa järjestelmän toiminta ruuhkahuipun aikana ja liiketoimintaa kuukaudessa suoritettujen transaktioiden kokonaislukumäärä. Näiden lukujen pohjana voi sekä kehittäjällä että liiketoiminnan edustajalla olla mittarina samat järjestelmästä mitatut arvot. Mittareiksi kelpaavia arvoja voi olla esimerkiksi vasteaika, resurssin käyttöaste tai järjestelmän osan läpimenoaika. Mittausten tulee olla toistettavissa, luotettavia, helposti mitattavissa, johdonmukaisia ja itsenäisiä niin, että mittausten sidosryhmät eivät vaikuta itse tuloksiin. (Bondi 2015, Chapter 2)

Eräs yksinkertainen tapa mitata järjestelmän tehokkuutta on mitata kauanko jonkun tehtävä suoritus kestää. Mittaustuloksia kerätään toistamalla tätä tehtävää monta kertaa. Javan tekemä ajonaikainen kääntäminen aiheuttaa pienen hankaluuden tässä mittaustavassa. Javassa koodin käännöksen optimointi saattaa kestää minuutteja, ja tämä tulee ottaa testatessa huomioon. Javalla kirjoitetun järjestelmän tehokkuutta mitatessa on tärkeää järjestää testiympäristö niin, että järjestelmää ei testata kylmiltään. Järjestelmää tulee siis käyttää ensin jonkin aikaa ja vasta sen jälkeen aloittaa toiminnan mittaaminen. (Oaks 2014, 24.)

Opinnäytetyössä analysoin järjestelmälle uudenlaista toimintaa ja selvitin yleisesti laajennuksen ja järjestelmän toiminnan tehokkuutta mittaamalla toimintaa. Toiminnallinen konteksti analyysissani on yrityksen järjestelmän ja sen laajennoksen toiminta vedonlyöntipörssissä vinttikoirakilpailuissa. Kontekstin ajoympäristöksi valitsin nykyjärjestelmän tuotantoympäristön. Järjestelmän toiminnan tehokkuuden mittaustavaksi valitsin mitata vinttikoirakilpailuun liittyvää toiminallisuutta ja toistaa tätä. Tein laajennokselle toiminalliset perustestit ja kokeilin toimintaa kahdessa vinttikoirakilpailukohteissa. Perustestien ja kokeilun jälkeen pystyin toteamaan, että laajennos toimi järjestelmän muiden osien kanssa yhtäaikaisesti, joten olin valmis siirtymään mittaamaan toimintaa. Tein mittaukset ns. aidolla järjestelmällä, eli yrityksen tuotantoympäristössä (ns. macrobenchmarks), saadakseni mittauksista mahdollisimman realistisia tuloksia. Tätä kautta saisin parhaiten tietoa siitä, miten nopeasti tai hitaasti tekemäni laajennos toimii ja miten muu järjestelmä toimii sen mukana. Koska kyseessä on Javalla kirjoitettu järjestelmä, tuli testatessa ottaa myös huomioon ajonaikainen kääntäminen. Käyttäessäni testaamiseen tuotantoympäristöä, joka oli käynnissä koko ajan, saatoin olla varma, etten tulisi testanneeksi järjestelmää kylmiltään jolloin tulokset eivät välttämättä kuvaisi oikeaa toimintaa. Mittasin laajennoksen toimintaa tallentamalla siihen liittyvien osasuoritusten suoritusaikoja ja laskemalla niistä avainlukuja, mm. Keskiarvosuoritusaikoja ja mediaanisuoritusaikoja. Koska ne ovat hyviä vertailulukuja, ovat ne erityisen käyttökelpoisia, kun järjestelmään tehdään muutoksia. Järjestelmän keskiarvo- tai mediaanisuoritusaikojen parantuessa voidaan todeta tehtyjen muutosten ol-

leen toimivia. Toisaalta näiden aikojen hidastuessa, voidaan todeta järjestelmässä tapahtuneen merkittävää tehokkuuden heikkenemistä. Mittauksia varten syötin järjestelmään 15 vinttikoirakilpailua ja näissä kilpailuissa suoritettiin yhteensä 249 kertaa kertoimien päivitystoiminallisuus.

Valitsin mitattavaksi toiminnallisuudeksi kertoimien päivitystoiminallisuuden. Toiminallisuus on usein toistettavissa sekä mitattavissa ja sen tehokkuus on kriittinen osa järjestelmän laajennuksen toimintaa. Päivitystoiminallisuudessa algoritmimoduulille lähetetään vedonlyöntikohteessa päivittyneet tiedot. Moduuli laskee kilpailulle tietojen perusteella uudet kertoimet ja jättää päivittyneet kertoimet haettavaksi. Päivittyneiden kertoimien perusteella jätetään uudet päivittyneet vedot uusine kertoimineen vedonlyöntipörssiin. Jaoin tämän toiminnon osasuorituksiin, joiden suoritusaikoja analysoin mittaamalla niiden avainlukuja. Ensisijaisesti halusin selvittää, toimiiko nykyinen järjestelmä riittävän nopeasti käytettäväksi vinttikoirakilpailujen kanssa ja toissijaisesti löytää suorituksesta tähän tarkoitukseen liian hitaita osatoimintoja. Analyysia varten kertoimien päivityksen suorittaminen jaettiin 14 osasuoritukseen. Osasuoritukset valittiin mahdollisimman pieninä mutta kuitenkin merkittävän kokoisina osina päivitysprosessia. Analysoinnissa lähteenä käytettiin erillistä lokitiedostoa, josta löytyi osasuoritusten alkamisajankohdat ja päättymisajankohdat.

7.1 Osasuoritusten mittaaminen

Järjestelmän kirjoittamat lokirivit osasuoritusten alkamisesta ja loppumisesta luotiin käynnissä olevasta tuotantojärjestelmästä. Järjestelmä kirjoitti lokiin suorituksia kaikista samaan aikaan käynnissä olevista kohteista ja niiden osasuorituksista, myös kohteista jotka eivät olleet tätä analyysia varten kiinnostavia. Nämä olivat järjestelmässä käynnissä olevia muita pelikohteita kuin vinttikoirakilpailuja. Analyysia varten tehty lokitiedosto (liite 1) oli liian suuri käsin tarkasteltavaksi, joten kirjoitin analysointia varten Java-ohjelman LogAnalyzer.java (liite 2). LogAnalyzerin tulisi lukea lokitiedosto, laskea osasuoritusten kestoista tunnuslukuja ja tulostaa tunnusluvut tarkempaa analyysia varten. Analyysia varten kirjoittamani LogAnalyzer ohjelmapätkään kirjoitin yksikkötestit jUnit-kirjastolla. (Liite 3). Yksikkötestauksella pystyin saamaan kohtalaisen varmuuden ohjelman toiminnasta. Yksikkötestauksella sain esiin toiminnallisia virheitä ohjelmasta ja lopulta riittävän varmuuden siitä, että sillä lasketut luvut ovat vähintään oikeansuuntaiset. Täydellistä varmuutta ei testeillä saatu, mutta tähän tarkoitukseen totesin ohjelman toimivan riittävällä varmuudella oikein.

Käynnistäessäni vinttikoirakilpailukohteita, otin talteen näiden kohteiden vedonlyöntipörssistä saadut id:t samalla kun syötin ne järjestelmään. Myöhemmin annoin nämä id:t syötteeksi LogAnalyzerille, joka tunnisti marketId-kentän avulla oikeat kohteet lokitiedostosta. Tunnistetuista riveistä LogAnalyzer laski yhden säikeen suoritukseen käyttämän ajan millisekunteina osasuorituksen alku- ja loppurivien kellonaikakentästä. Yksittäiset ajat talletettiin muistiin HashMap-tietorakenteeseen, jossa luvut oli ryhmitelty osasuorituksittain. Osasuoritukset iteroitiin ja niistä laskettiin valittuja tunnuslukuja. Tunnusluvuiksi pyrin valitsemaan sellaisia mitattavia tunnuslukuja, jotka oleellisesti kertoisivat suorituksen toiminnasta, olisivat toistettavissa ja siten uudelleen mitattavissa sekä olisivat luotettavia.

Valitsin tunnusluvuiksi keskiarvon, maksimin, mediaanin sekä yhteensä suoritukseen käytetyn ajan millisekunteina. Keskiarvo ja mediaani arvot kertovat miten osasuoritus toimii yleisesti ja minkä verran aikaa ne vievät kokonaissuorituksesta. Osasuorituksen maksimi arvon poiketessa reilusti keskiarvosta tai mediaanista, on se merkki, että järjestelmässä on tapahtunut poikkeava tilanne, joka tulee tutkia tarkemmin. Kokonaissuoritusaikaa tutkimalla voisi analysoida millainen vaikutus kokonaisuuden nopeuttamiseen osasuorituksen parantamisella olisi. Osasuoritus jolla olisi pieni mediaani suoritusaika, mutta suuri kokonaiskesto voisi olla potentiaalinen korjauskohde, sillä pienikin parannus voisi vähentää merkittävästi kokonaiskestoa.

Lopuksi LogAnalyzer tulosti päivitysprosessin osasuorituksien tunnusluvut CSV-muodossa. Tästä tulostuksesta mittarit oli helppo poimia Exceliin käsittelyä varten, jossa tietoja oli helpompi tarkastella ja siistiä. Järjestin Excelissä osasuoritukset itselleni helpoimmin ymmärrettävään muotoon, pyöristin liukuluvut yhteen desimaaliin sekä siistin taulukon. Excelissä taulukon lukuja oli helppo tutkia, ja valita mitkä olisivat epänormaalin pitkiä suoritusaikoja, joita kannattaisi tutkia tarkemmin.

Taulukossa 3 näkyvät osasuorituksen nimi, montako kertaa suoritus on testiaineistossa tehty, suorituksen maksimikestoaika, suorituksen keskiarvoaika, suorituksen mediaaniaika ja yhteensä suoritukseen käytetty aika. Taulukon kaikki ajat ovat millisekunneissa.

Taulukko 3 Osasuoritusten suoritusajat

Osasuoritus	lkm	Maksimi	Keskiarvo	Mediaani	Yhteensä
cleanActiveBets	270	10560	758	677	204625
performBetCancellation	258	806	408	440	105315
foundMatch	249	10948	324	70	102658
getAvailableOffers	268	465	120	117	32216
saveOrMarket	271	2951	108	2	29211
cancelRiskBets	270	372	47	0	12633
getMarketFromCache	271	915	23	3	6205
updateProbabilities	270	106	2	1	515
updateProbDeserialize	270	106	2	1	471
lockMarket	271	7	1	0	152
createBets	268	7	0	0	49
storeReporting	258	4	0	0	22
truncateBet	268	2	0	0	20
clearAmount1	270	6	0	0	13

7.2 Mittaustulosten analysointi ja havaitut ongelmat

Halusin analysoidessa havaita, onko yrityksen järjestelmän nykyarkkitehtuurissa tai ohjelmiston osissa sellaisia osia, jotka eivät toimi riittävän nopeasti vinttikoirakilpailuissa. Selkeitä pullonkauloja tai toiminallisia ongelmakohtia ei järjestelmästä löytynyt vaan kaikki vaikutti toimivan kohtalaisen tehokkaasti. Osa mittausten tunnusluvuista vaikutti kuitenkin olettamaani huonommilta. Vedonlyöntipörssin markkinoissa nopea reagointi hinnanmuutoksiin on tuloksen teon kannalta kriittistä, joten pienetkin nopeutukset järjestelmässä voisivat olla merkittäviä. Tutkin tarkemmin mikä tai mitkä osasuorituksista olisivat sellaisia joissa voisi olla tehostamisen varaa. API-kutsut ulkopuolisiin rajapintoihin verkon yli hidastavat aina toimintaa ja niiden määrän vähentämisellä tai nopeuttamisella voisi olla merkittävää parannusta järjestelmän nykytoimintaan.

Löytääkseni potentiaalisia parannuskohteita tutkin taulukon tunnusluvuista mikä tai mitkä suorituksista näyttivät kutsulukumäärältään, keskiarvoltaan tai yhteensä käytetyltä ajalta siltä, että niiden toimintaa tulisi tutkia tarkemmin. Valitsin tarkemmin analysoitaviksi osasuorituksiksi:”cleanActiveBets”, koska se oli käyttänyt yhteensä eniten suoritusaikaa, ”foundMatch” ja ”getMarketFromCache”, koska niillä oli epäilyttävän suuria maksimisuoritusaikoja, ja ”performBetCancellation” koska se oli käyttänyt merkittävän osan yhteensä suoritusajasta.

FoundMatch-suoritus kuvaa tapahtumaa pelatun vetotarjouksen havaitsemisesta siihen, että uusien kertoimien laskenta järjestelmässä aloitetaan. Suoritukseen käytetyt millisekunnit ovat aikaa, jonka tehtävä viettää järjestelmän sisäisissä jonoissa odottamassa suoritusvuoroaan. FoundMatch valikoitui analysoitavaksi, koska suorituksen aikana ei käytetä järjestelmän ulkoisia komponentteja, siihen käytettiin aikaa kokonaissuoritusajasta toiseksi eniten ja ennen kaikkea suuren maksimisuoritusajan takia. Hypoteesina oli voiko kyseessä olla oire järjestelmän rinnakkaissuoritusten takia tapahtuvasta ns. ”nälkiintymisestä”, jossa suoritus odottaa omaa suoritusvuoroaan.

Käytin Exceliä ja osasuoritusten kirjoittamaa lokitiedostoa sekä järjestelmän yleistä lokitiedostoa avuksi, selvittääkseni tarkasti mitä järjestelmässä oli tapahtunut silloin kun osasuoritukseen oli käytetty aikaa yli 2000ms. Löysin mittausaineistosta muutaman virhehavainnon, jossa havainto oli tehty juuri kun kilpailu oli alkanut. Tämä oli aiheuttanut molemmissa tapauksissa (virheellisen) pitkän suoritusajan, joka näkyi tuloksissa pitkinä jonotusaikoina. Nämä kuitenkaan eivät olleet järjestelmän todellisia suoritusaikoja vaan johtuivat LogAnalyzerin virheellisestä toiminnasta, joten FoundMatch-suorituksen osalta saatoin jättää kehitysehdotukset tekemättä. Päädyin kuitenkin ehdottamaan järjestelmän lokituksen parantamista, jotta pitkittyneet suoritusajat olisi helpompi havaita sekä matalalla prioriteetilla tutkimaan voiko kilpailun alkamisen jälkeen tehtyjä API-kutsuja keskeyttää nopeammin kuin havaitsemani nykyinen vajaa 9 sekuntia. Otettuani tilastoista pois pitkittyneet suoritusajat, putosi foundMatch-osasuorituksen keskiarvo 324 millisekuntiin, joka ei enää vaikuttanut ongelmallisen suurelta ajalta osasuoritukselle.

CleanActiveBets-suoritus kuvaa tapahtumaa, jossa järjestelmä vertaa omia tietojaan vedonlyöntipörssin tietoihin ja päivittää järjestelmän sisäisiä tietoja vastaaviksi. CleanActiveBets-suoritus valikoitui analysoitavaksi, sillä se käytti eniten kokonaissuoritusaikaa. Aloitin analysoinnin selvittämällä syyn suurimpiin, mediaanista huomattavasti poikkeaviin suoritusaikoihin. Poikkeavan suuria suoritusaikoja löytyi testiaineistosta lopulta vain kaksi kappaletta. Koska niissä kulunut aika oli käytetty vedonlyöntipörssiin tehdyssä API-kut-

sussa, ei niiden osalta kehitysehdotuksia yrityksen järjestelmään löytynyt. Näiden suurimpien suoritusaikojen poistaminen tutkittavasta materiaalista, ei tehnyt merkittävää vaikutusta kokonaissuoritusajan vähentymiseen. Kokonaissuoritusaika koostui suuresta määrästä kutsuja (270 kpl) keskiarvokutsun ollessa järjestelmän ulkopuolisen API-kutsun takia merkittävän suuri (756ms). Kehitysehdotuksena kirjasin kutsun suorittamisen harvemmin, jotta kokonaissuoritusaika saadaan pienentymään merkittävästi. Kirjasin kehitysehdotuksena myös, että osasuorituksen voisi vaihtoehtoisesti jatkossa ajaa erillään ajastetusti jolloin se ei hidastaisi enää päivitysprosessia.

GetMarketFromCache-suoritus kuvaa tapahtumaa, jossa järjestelmän välimuistina toimivasta Redis-tietokannasta haetaan pelikohteeseen liittyvä tietorakenne. GetMarketFromCache-osasuoritus valikoitui analysoitavaksi epäilyttävän pitkien maksimisuoritusaikojen takia. Redis on nopea muistitietokanta, ja koodissa tapahtuva toiminta ei pitäisi olla lainkaan pitkäkestoinen. Välimuistikomponentin kanssa toimiva koodi käsittelee kuitenkin järjestelmän sisäisiä lukkoja, ja poikkeavan pitkät suoritusajat voisivat vaikuttaa ongelmilta järjestelmän lukituksissa.

Testiaineistosta löytyi 7 kpl suorituksia, joiden kestoaika poikkesi selvästi mediaanista (6ms). Suoritusten aikana järjestelmässä on ollut muuta merkittävää kuormaa jotka käyttävät välimuistia. Tutustuttuani tarkemmin välimuistin toimintaan, havaitsin, että osasuorituksessa käytetyt kutsut lukitsivat tarpeettomasti koko välimuisti-luokan. Tällöin vain yksi kutsu kerrallaan pystyi käyttämään välimuistiluokan palveluita.
Kirjoitin korjauskehotukseksi tutkia ja varmistaa, tarvitaanko välimuistin metodeissa lukita koko luokkaa, vai voisiko vain luokan tietyt metodit lukita. Tämä nopeuttaisi melkoisella varmuudella välimuistin toimintaa näissä hitaimmissa tapauksissa, joissa sen toiminta hidastui jopa monisatakertaiseksi oletetusta. Kutsujen mitattu mediaani oli 3ms mutta havaittu maksimisuoritus aineistossa oli jopa 915ms.

PerformBetCancellation-suoritus kuvaa tapahtumaa, jossa järjestelmä peruuttaa vetotarjouksia vedonlyöntipörssistä. Suorituksen aika saattaa vaihdella suuresti tämän suorituksen kohdalla, koska suorituksessa otetaan ulkopuoliseen järjestelmään yhteyttä verkon yli. Suorituksen nopeuteen vaikuttaa järjestelmän sen hetkinen tila, käytettävissä olevat resurssit ja ulkopuolisen järjestelmän (vedonlyöntipörssi) tila. Halusin varmistua analysoidessani siitä, että järjestelmä toimii omalta osaltaan mahdollisimman hyvin saadakseen vedot peruutettua pörssistä mahdollisimman nopeasti. Tähän osasuoritukseen järjestelmä näytti käyttävän merkittävän osan kokonaissuoritusajasta, ja tällöin yksittäisen suoritusajan nopeuttamisella olisi merkittävä vaikutus kokonaissuorituksen kestoon.

Havaitsin, että aika näytti kuluvan vedonlyöntipörssiin tehtävän API-kutsun suorittamisessa. API-kutsua suoritettaessa luotiin aina uusi HTTP-yhteys, eikä ohjelmakoodissa käytetty lainkaan hyväksi HTTP-yhteyksien varastointia. Kirjasin parannusehdotukseksi kirjoittaa uudelleen HTTP-yhteyksien luonnin ja varastoinnin. Tämä nopeuttaisi osasuoritusta suurella varmuudella.

Analysoidessani ohjelmakoodia havaitsin myös muutamia yleisiä korjaus- /parannusehdotuksia. Vetotarjousten päivityksen päätyessä poikkeustilanteeseen ei lokiin kirjoitettu poikkeusta lainkaan, joka hankaloitti virhetilanteen selvittämistä. Poistin järjestelmästä ns. kuollutta koodia, jota mikään osa ei kutsunut. Lisäksi dokumentoin järjestelmän toimintaa koodin sekaan selvittäessäni sitä.

7.3 Havaittujen ongelmakohtien korjaus

Opinnäytetyön alkuperäinen tarkoitus oli testata järjestelmän toimintaa ja havaita mittausten avulla mahdollisia pullonkauloja järjestelmän suorituksessa. Analyysia tehdessäni olin kuitenkin paneutunut niin syvällisesti osasuoritusten toimintaan, että katsoin useassa kohdassa järkeväksi myös samalla toteuttaa kirjaamiani parannusehdotuksia ja korjata niiden osalta järjestelmän nykytoimintaa tehokkaammaksi.

” CleanActiveBets ”-osasuoritus oli hidas, sillä siinä tehtiin useita API-kutsuja vedonlyöntipörssiin, ja se aiheutti merkittävää hidastusta koko vetojenpäivitysprosessiin. Koska suoritus ei ollut merkittävä vetojenpäivitysprosessissa, poistin sen suorittamisen tästä osasta kokonaan. Korjauksena kirjoitin järjestelmään uuden eräajon, joka iteroi kaikki järjestelmässä olevat pelikohteet läpi, ja suoritti niille ajastetusti yksi kerrallaan tämän toiminallisuuden riittävän usein. Korjauksen jälkeen osasuoritus ei enää hidastanut vetojen päivitystä, vaan se toimi omana itsenäisenä järjestelmän osana.

Osasuoritus ”getMarketFromCache” aiheutti hidastelua välimuistiluokan lukitusten takia. Ohjelmakoodin päivitys vaati huolellista koodin lukemista, ja päädyin lopulta tämän osalta vain poistamaan yhdestä hyvin usein käytettävästä metodista ”synchronized”-sanan. Metodi oli sama mitä getMarketFromCache-osasuoritus käytti. Synchronized-sanan käyttö metodissa aiheuttaa Java-kielessä, että luokkaan otetaan lukko metodi-kutsun ajaksi. Lukituksen aikana muita ”synchronized” merkittyjä metodeita ei voida kutsua. Vaikutti siltä, että muiden ”synchronized”-metodien käyttö luokassa hidasti GetMarketFromCache-metodin toimintaa. Korjauksen validointi tulisi kuitenkin huomata mittausten avulla.

Osasuorituksen ” performBetCancellation ” toiminta oli hidasta, sillä vedonlyöntipörssin APIa kutsuttaessa luotiin joka kerta uusi HTTP-yhteys eikä hyväksikäytetty nk. yhteyksien varastointi-tekniikkaa. Kirjoitin uusiksi yhteyksien luonnin käyttämällä yleisesti tunnettua Apachen PoolingHTTPClientConnectionManager-kirjastoa. Korjauksen jälkeen HTTP-yhteys luotiin kutsua varten uudelleen vain, kun yhteys oli todella katkennut, eikä enää joka kerta kutsua tehdessä.

7.4 Korjausten vaikutus toimintaan

Ymmärtääkseni paremmin tekemieni korjausten vaikutuksen yrityksen järjestelmän toimintaan, ajoin järjestelmän uudessa versiossa vinttikoirakilpailuja. Ajaessani mittaisin osasuoritukset samalla tavalla kuin aiemmin ja vertaisin suoritusajoissa tapahtuneita muutoksia. Pyrin suorittamaan mittaukset niin samankaltaisina kuin aiemmin, jotta mitattavat luvut olisivat vertailukelpoisia. Osasuoritusaikojen ollessa tällä mittauksella merkittävästi lyhyempiä, voisin varmistua tekemieni korjausten positiivisesta muutoksesta tehokkuuteen. Tässä uudelleen ajamassani aineistossa tapahtui 365 kertoimien päivitystoiminnallisuutta. Osasuorituksiin käytettyjen aikojen lisäksi laskin mediaanisuoritusajat koko päivitysprosessista ennen korjauksia ja korjausten jälkeen. Päivitysprosessin mediaanisuoritusaika ennen korjauksia oli 1358ms, ja tekemieni korjausten jälkeen enää 316ms. Kokonaisuutena korjaukset olivat tehneet ison parannuksen vasteaikaan.

Osasuoritusten suoritusajat paranivat käytännössä kaikissa osasuorituksissa. On tärkeää huomioida myös, että minkään osasuorituksen suoritusaika ei merkittävästi heikentynyt, eli korjaukset eivät olleet aiheuttaneet toiminnalle vahinkoa.

Taulukkoon 4 on kirjattu korjausten jälkeen ajettujen vinttikoirakohteiden osasuoritusten kestot millisekunneissa, sekä muutoksen suuruus millisekunneissa verrattuna ennen korjauksia tehtyihin mittaustuloksiin.

Taulukko 4 Osasuoritusten kestoajat korjausten jälkeen

Osasuoritus	lkm	Maksimi	Max muutos	Keskiarvo	Ka muutos	Mediaani	Mediaanin muutos
cancelRiskBets	70	189	-183	7	-39	0	0
clearAmount1	181	3	-3	0	0	0	0
createBets	178	1	-6	0	0	0	0
FoundMatch	143	365	-1350	52	-273	41	-27
getAvailableOffers	75	466	1	28	-92	20	-97
getMarketFromCache	183	36	-879	3	-20	2	-1
lockMarket	183	9	2	0	0	0	0
performBetCancellation	65	406	-400	217	-191	226	-214
saveOrMarket	150	249	-2702	10	-98	2	0
storeReporting	156	1	-3	0	0	0	0
truncateBet	178	1	-1	0	0	0	0
updateProbabilities	181	8	-98	2	0	1	0
updateProbDeserialize	181	8	-98	2	0	1	0

Oletin että käyttäessäni yhteyksien varastointi-tekniikkaa, olisivat vedonlyöntipörssiin tehtyjen kutsut aiempaa nopeampia. Osasuorituksen performBetCancellations mediaaniaika väheni 214 millisekuntia, alkuperäisen ollessa 440ms ja nyt 226ms. Osasuorituksen getAvailableOffers käytetty mediaaniaika väheni 97ms alkuperäisen ollessa 117ms ja nyt 20ms. Molemmat ovat todella merkittäviä nopeutuksia päivitysprosessin ja koko järjestelmän toiminallisuuteen. Vinttikoirakilpailumittausten ulkopuolelta havaittiin myös, että HTTP-yhteyksiä uudelleenkäyttämällä saavutettiin merkittävää parannusta järjestelmän suorituskykyyn myös muualla kuin tässä päivitysprosessissa.

Osasuorituksen GetMarketFromCache maksimisuoritusajat olivat olleet aiemmin suuria, jopa 915ms. Nyt kun lukitus oli poistettu, saatiin maksimisuoritusajaksi vain 36ms. Myös toisen välimuistia käyttävän osasuorituksen (SaveORMarket) maksimiaika pieneni muutosten jälkeen todella merkittävästi. SaveORMarket-osasuorituksen lukitukseen en tehnyt muutoksia sen tarpeellisuuden vuoksi. Suoritus kuitenkin nopeutui, koska moninkertaisesti useammin käytetty GetMarketFromCache ei enää varannut luokan lukkoa jatkuvasti.

Molempien suuret maksimiaika-tulokset olivat yksittäisiä suorituksia, joissa suorittava säie ei ollut saanut lukittua välimuisti-luokkaa itselleen järjestelmän ruuhkautuneisuuteen vuoksi. Kokonaisuutena ylimääräisen lukituksen poistaminen oli todella merkittävä parannus järjestelmän toimintaan.

Lisäksi oli havaittavissa, että myös muut avainluvut olivat parantuneet: foundMatch ja cancelRiskBets suoritusten maksimiaika olivat pudonneet merkittävästi. Päivityksen kokonaissuorituksen keskiarvosuoritusaika väheni muutosten jälkeen 1793 millisekunnista 332 millisekuntiin. Tärkeää on huomata, että minkään osasuorituksen suoritusaika ei kasvanut merkittävissä määrin, järjestelmä vaikuttaisi toimivan kaikilta osin yhtä hyvin tai paremmin kuin aiemmin.

Tuloksia tarkastellessa tulee kuitenkin huomioida, että järjestelmän muulla tilalla on toimintaan merkitystä eikä lukuarvoja tule tuijottaa liian tarkasti. Merkitsevää tuloksissa oli muutosten suunta, joka antoi selkeän kuvan, että tehdyt korjaukset olivat tehostaneet järjestelmän toimintaa huomattavasti. Korjaukset, ja järjestelmän tehostunut toiminta auttaa yritystä laajentamaan toimintaa vedonlyöntipörsseissä ja muualla.

7.5 Yrityksen antama palaute työstäni

Yritys oli opinnäytetyöhöni ja siinä tehtyihin parannuksiin erittäin tyytyväinen. Tekemäni korjaukset vaikuttivat koko nykyjärjestelmään, ja paransivat yleisesti yrityksen palvelun toimintaa. Sain työhöni seuraavaa palautetta:

> *Jorma Tennin opinnäytetyö oli meille suureksi hyödyksi. Työn taustalla oli kiinnostus selvittää palvelumme käyttömahdollisuudet vinttikoirakilpailuissa ja mitä kehitystyötä tämä vaatisi. Jorma Tenni teki tarvittavat tekniset kehitystyöt nopeasti ja pystyi laajentamaan työtään tehostaakseen palveluamme vastaamaan vinttikoirakilpailujen vaatimuksia. Työn tuloksena havaittiin, että Jorman tekemillä muutoksilla järjestelmän nykyinen arkkitehtuuri toimii, ja palvelumme pystyy teknisesti toimimaan tässä ympäristössä. Työtä seuranneissa testeissä kävi kuitenkin ilmi, että lähtökerroinanalyysimme oli odotettua heikompaa ja että tällä saralla yrityksen pitää tehdä lisää kehitystyötä. Tämä kehitystyö on matemaattistilastollinen ja selkeästi tämän opinnäytetyön ulkopuolella.*
>
> *Tennin työ kehitti meidän palveluamme merkittävästi ja sisälsi useamman hyvän huomion ja johtopäätöksen. Arvokkain muutos oli vasteaikamme moninkertainen pudotus.*

Tästä palvelun tehostamisesta on ollut paljon hyötyä myös muissa tilanteissa, joissa tapahtuu yhtäkkisiä muutoksia kohteen todennäköisyyksiin. Vasteajan lyhentäminen on myös tärkeä kehitysaskel kohti in-play vedonvälityspalvelua, joka on yrityksellemme strategisesti erittäin tärkeä.

Olof Stenius, toimitusjohtaja

8 Yhteenveto

Työtä aloittaessani yritys haki kasvumahdollisuuksia etsimällä uusia käyttökohteita kerroinpäivityspalvelulle ja sitä varten kehitetyille algoritmeille. Vedonlyöntipörssistä uudeksi käyttökohteeksi valikoitui vinttikoirakilpailukohteet, niiden suuren päivittäisen lukumäärän tarjotessa suurta potentiaalia laajalle toiminnalle. Oli epävarmaa toimisiko nykyinen järjestelmä sellaisenaan nopeatempoisissa vinttikoirakilpailukohteissa, vai vaatisiko laajentaminen suuria muutoksia nykyjärjestelmään. Lähdin tekemään ensin laajennusta ja sitten mittaamaan laajennuksen toimintaa.

Vaikka laajennoksen vaativien koodimuutosten tekeminen oli melko nopeata, vei koodin lukeminen, ymmärtäminen ja mittaustulosten analysointi oman aikansa. Järjestelmän toimintaa mitattuani ja mittauksia analysoidessani löysin useita kohtia järjestelmästä, jotka toimivat oikein, mutta joiden suoritusta pystyi tehostamaan merkittävästi pienillä koodimuutoksilla. Tällaisia pienet toimintaa tehostavat parannukset ovat usein työläitä, koska niiden tekemiseksi tulee tehdä paljon analysointia, jotta ymmärtää miksi toiminta on nyt hidasta ja miten sen saisi korjattua nopeammaksi. Usein tällaiset korjaukset jäävät tekemättä päivittäisessä työssä ajanpuutteen takia.

Opinnäytetyötä tehdessäni vahvistui kokemuksen kautta saamani oppi, että ohjelmistokoodin uudelleenkirjoittaminen parantaa merkittävästi ohjelmiston toimintaa. Ohjelmistokoodia kirjoittaessa, ohjelmoija kirjoittaa koodin jotain tiettyä tarkoitusta varten. Lähtökohtaisesti kirjoitettavan koodin halutaan olla uudelleenkäytettävä, mutta koodia uudelleen käytettäessä alkuperäinen koodin tarkoitus ei välttämättä ole enää merkittävä tai edes tiedossa. Koodin toimiessa silti oikein, ei sen tehokkuuteen välttämättä osata tai huomata kiinnittää huomiota. Opinnäytetyötä tehdessäni tehostin järjestelmän toimintaa uudelleenkirjoittamalla toimivaa koodia, jossa ohjelman sisäiset lukitukset eivät olleet enää merkityksellisiä vaan hidastivat toimintaa.

Opinnäytetyössäni onnistuin tavoitteiden mukaisesti laajentamaan järjestelmän toimintaa, sekä selvittämään ja korjaamaan useita järjestelmässä olleita hitaita ratkaisuja. Yritys sai

paremman ymmärryksen järjestelmän nykytilasta ja toiminnasta. Yritys oli tekemääni työhön tyytyväinen, vaikka toisaalta havaittiin, että vinttikoirakilpailuja varten tarvitaan lisää kehitystyötä paremman lähtökerroinanalyysin tekemiseksi. Opin työssäni vedonlyönnin alan toiminnasta, vedonlyöntipörssien toiminnasta, vedonlyöntimarkkinoiden käyttäytymisestä vedonlyöntipörsseistä sekä laajensin ymmärrystäni IT-järjestelmien toiminnan tehokkuuden mittauksesta.

Lähteet

Britannica 2018. Britannica verkkojulkaisu.
Luettavissa https://www.britannica.com/sports/dog-racing
Luettu 17.3.2019

Bondi, Andre B. 2015. Foundations of software and system performance engineering: process, performance modeling, requirements, testing, scalability, and practice. Addison-Wesley.

Business Insider 2016. Business Insider Nordic verkkojulkaisu.
Luettavissa https://nordic.businessinsider.com/smarkets-betting-startup-reports-160-increase-in-annual-profits-2017-7?r=UK&IR=T
Luettu 17.03.2019

Egba 2013. Sports Betting Report.
Luettavissa http://www.egba.eu/media/Sports-Betting-Report-FINAL.pdf

Fortier, Paul J., Michel, Howard Edgar. 2003.Computer Systems Performance Evaluation and Prediction. Digital Press

Gambling Commission 2018. Gambling Commission verkkojulkaisu.
Luettavissa https://www.gamblingcommission.gov.uk/news-action-and-statistics/Statistics-and-research/Statistics/Industry-statistics.aspx
Luettu 17.3.2019

Kucharski, A. 2016. The Perfect Bet. Profile Books.

Oaks, S. 2014. Java Performance: The Definitive Guide. O'Reilly Media Inc.

Pinnacle 2019. Pinnaclen verkkosivu.
Luettavissa https://www.pinnacle.com/fi/betting-articles/educational/profitable-betting-from-chance-to-choice/SCSJG6UB3GE388XT
Luettu 23.3.2019

Veikkaus 2017. Veikkaus vuosiraportti.
Luettavissa https://cms.veikkaus.fi/site/binaries/content/assets/dokumentit/vuosikertomus/2017/veikkaus_vuosiraportti_2017.pdf
Luettu 17.3.2019

Vuoksenmaa, Jorma. 2016. Pelaajalta Pelaajille. Unibet Group.

Williams, Leighton Vaughan. 2004. Betting to win. High Stakes.

Yao, King. 2008. Weighing the odds in sports betting. Pi Yee Press.

Liitteet

Liite 1. Ote osasuoritusten lokitiedostosta

2018-10-10 07: 26.908 [pool-2-thread-13] - MarketId 1.149276330 cleanActiveBets() started.
2018-10-10 07: 26.935 [pool-2-thread-25] - MarketId 1.149180986 getAvailableOffersExcludingOwnBets() done.
2018-10-10 07: 26.935 [pool-2-thread-25] - MarketId 1.149180986 createBetsAndCancellations() started.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 createBetsAndCancellations() done.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 truncateBetOnMaxAllowedLiability() started.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 truncateBetOnMaxAllowedLiability() finished.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 performBetCancellationsAndBetPlacements() started.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 performBetCancellationsAndBetPlacements() finished.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 storeReporting started.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 storeReporting done.
2018-10-10 07: 26.936 [pool-2-thread-25] - MarketId 1.149180986 saveORMarket() started.
2018-10-10 07: 26.937 [pool-2-thread-25] - MarketId 1.149180986 saveORMarket() done.
2018-10-10 07: 26.938 [pool-2-thread-25] - MarketId 1.149180986 unlockMarketLock() finished.
2018-10-10 07: 27.809 [pool-2-thread-13] - MarketId 1.149276330 cleanActiveBets() done.
2018-10-10 07: 27.809 [pool-2-thread-13] - MarketId 1.149276330 getAvailableOffersExcludingOwnBets() started.
2018-10-10 07: 27.928 [pool-2-thread-13] - MarketId 1.149276330 getAvailableOffersExcludingOwnBets() done.
2018-10-10 07: 27.929 [pool-2-thread-13] - MarketId 1.149276330 createBetsAndCancellations() started.
2018-10-10 07: 27.929 [pool-2-thread-13] - MarketId 1.149276330 createBetsAndCancellations() done.
2018-10-10 07: 27.929 [pool-2-thread-13] - MarketId 1.149276330 truncateBetOnMaxAllowedLiability() started.
2018-10-10 07: 27.929 [pool-2-thread-13] - MarketId 1.149276330 truncateBetOnMaxAllowedLiability() finished.
2018-10-10 07: 27.929 [pool-2-thread-13] - MarketId 1.149276330 performBetCancellationsAndBetPlacements() started.
2018-10-10 07: 28.330 [pool-2-thread-13] - MarketId 1.149276330 performBetCancellationsAndBetPlacements() finished.
2018-10-10 07: 28.330 [pool-2-thread-13] - MarketId 1.149276330 storeReporting started.
2018-10-10 07: 28.330 [pool-2-thread-13] - MarketId 1.149276330 storeReporting done.

Liite 2. Lokitiedoston parsintaan tehty Java-ohjelman lähdekoodi

```
package com.joppe;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.format.DateTimeFormatter;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.stream.Stream;

import com.google.common.collect.Lists;
import com.google.common.collect.Maps;
import com.google.common.collect.Sets;

/**
 * Class that parses Logback log-files written by OR-system / Greyhound module
 * while placing bet offers in exchange.
 *
 * Each line in file is read into an object LogLine and stored in list
 * fileAsLogLine. List is iterated and milliseconds between action started and
 * finished is stored in Map actionsTook. Finally actionsTook is printed to
 * console.
 *
 * @author tennijo
 *
 */
public class LogAnalyzer {

	public enum LogActionType {
		FOUND_MATCH_START, FOUND_MATCH_END, LOCK_MARKET_START, LOCK_MARKET_END, GET_MARKET_FROM_CACHE_START,
GET_MARKET_FROM_CACHE_END, UPDATE_PROBABILITIES_START, UPDATE_PROBABILITIES_END, UPDATE_PROBABILITIES_DESERIALIZE_START, UPDATE_PROBABILITIES_DESERIALIZE_END,
CLEAR_AMOUNT_1_START, CLEAR_AMOUNT_1_END, CLEAR_AMOUNT_2_START, CLEAR_AMOUNT_2_END, CANCEL_RISK_BETS_START, CANCEL_RISK_BETS_END, CLEAN_ACTIVE_BETS_START,
CLEAN_ACTIVE_BETS_END, GET_AVAILABLE_OFFERS_START, GET_AVAILABLE_OFFERS_END, CREATE_BETS_START, CREATE_BETS_END, TRUNCATE_BET_START, TRUNCATE_BET_END, PER-
FORM_BET_CANCELLATION_START, PERFORM_BET_CANCELLATION_END, STORE_REPORTING_START, STORE_REPORTING_END, SAVE_ORMARKET_START, SAVE_ORMARKET_END
	}

	String fileName = "/Users/tennijo/greyhound2.log";
	// This is used to filter out only greyhound marketIds from logfile which
	// contains data from all currently active markets.
	HashSet<String> greyhoundMarketIds = Sets.newHashSet("1.149276330", "1.149263095", "1.149262079", "1.149263099",
			"1.149317005", "1.149314488", "1.149316790", "1.149317007", "1.149580506", "1.149580595",
"1.149580553",
			"1.149580508", "1.149580597", "1.149580555", "1.149580510");

	Map<String, List<Long>> actionsTook = Maps.newHashMap();
	List<LogLine> fileAsLogLine = Lists.newArrayList();

	public void processLine(String line) {

		// replace non-breaking space with whitespaces
		line = line.replace('\u00A0', ' ');

		int indexOfMarketStart = line.indexOf(" 1.");
		// This is a MB line and not relevant for analysis
		if (indexOfMarketStart < 0) {
			return;
		}

		// parse marketId
		String substringFromMarketId = line.substring(indexOfMarketStart).trim();
		String marketId = substringFromMarketId.substring(0, substringFromMarketId.indexOf(' '));
```

```
                // parse datettime
                String dateString = line.substring(0, line.indexOf("[")).trim();
                DateTimeFormatter ofPattern = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSS");
                LocalDateTime parse = LocalDateTime.parse(dateString, ofPattern);

                // parse threadname
                String threadName = line.substring(line.indexOf("[") + 1, line.indexOf("]"));

                // parse message
                String message = substringFromMarketId.substring(substringFromMarketId.indexOf(marketId) + marketId.length())
                                                                .trim();

                LogLine ll = new LogLine(parse, threadName, marketId, message, getLogActionType(message));
                fileAsLogLine.add(ll);
        }

        public void analyze() {

                // Parse file to a arraylist of LogLines
                try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
                        stream.forEachOrdered(this::processLine);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                System.out.println("Parsed file to " + fileAsLogLine.size() + " objects");

                // Calculate times for parseFoundMatch, this is measured differently than all
                // others (as there are two different threads read...)
                parseFoundMatchTimes();
                // Calculate times for each operation and put to map

                parseStartEndKeyToMap(LogActionType.CANCEL_RISK_BETS_START, LogActionType.CANCEL_RISK_BETS_END,
                                                                "cancelRiskBets");
                parseStartEndKeyToMap(LogActionType.CLEAN_ACTIVE_BETS_START, LogActionType.CLEAN_ACTIVE_BETS_END,
                                                                "cleanActiveBets");
                parseStartEndKeyToMap(LogActionType.CLEAR_AMOUNT_1_START, LogActionType.CLEAR_AMOUNT_1_END, "clearAmount1");
                parseStartEndKeyToMap(LogActionType.CLEAR_AMOUNT_2_START, LogActionType.CLEAR_AMOUNT_2_END, "clearAmount2");

                parseStartEndKeyToMap(LogActionType.CREATE_BETS_START, LogActionType.CREATE_BETS_END, "createBets");
                parseStartEndKeyToMap(LogActionType.GET_AVAILABLE_OFFERS_START, LogActionType.GET_AVAILABLE_OFFERS_END,
                                                                "getAvailableOffers");
                parseStartEndKeyToMap(LogActionType.GET_MARKET_FROM_CACHE_START, LogActionType.GET_MARKET_FROM_CACHE_END,
                                                                "getMarketFromCache");

                parseStartEndKeyToMap(LogActionType.LOCK_MARKET_START, LogActionType.LOCK_MARKET_END, "lockMarket");

                parseStartEndKeyToMap(LogActionType.PERFORM_BET_CANCELLATION_START, LogActionType.PERFORM_BET_CANCELLATION_END,
                                                                "performBetCancellation");
                parseStartEndKeyToMap(LogActionType.SAVE_ORMARKET_START, LogActionType.SAVE_ORMARKET_END, "saveOrMarket");

                parseStartEndKeyToMap(LogActionType.STORE_REPORTING_START, LogActionType.STORE_REPORTING_END, "storeReporting");
                parseStartEndKeyToMap(LogActionType.TRUNCATE_BET_START, LogActionType.TRUNCATE_BET_END, "truncateBet");
                parseStartEndKeyToMap(LogActionType.UPDATE_PROBABILITIES_DESERIALIZE_START,
                                                                LogActionType.UPDATE_PROBABILITIES_DESERIALIZE_END, "updateProbDese-
rialize");

                parseStartEndKeyToMap(LogActionType.UPDATE_PROBABILITIES_START, LogActionType.UPDATE_PROBABILITIES_END,
                                                                "updateProbabilities");

                printMap();
        }

        private double getAverage(List<Long> values) {
                double sum = 0.0;
                for (Long long1 : values) {
                        sum += long1;
                }

                return sum / values.size();
        }
```

```
	private double getMin(List<Long> values) {
		double min = Double.MAX_VALUE;
		for (Long long1 : values) {
			if (long1 < min) {
				min = long1;
			}
		}
		return min;
	}

	private double getMax(List<Long> values) {
		double max = Double.MIN_VALUE;
		for (Long long1 : values) {
			if (long1 > max) {
				max = long1;
			}
		}
		return max;
	}

	private double getMedian(List<Long> values) {
		if (values.size() < 1) {
			return 0.0;
		}
		Collections.sort(values);
		if (values.size() % 2 == 0) {
			Long long1 = values.get(values.size() / 2);
			Long long2 = values.get(values.size() / 2 + 1);
			return (long1 + long2) / 2;
		}
		return values.get((values.size() / 2) + 1);
	}

	private long getSum(List<Long> values) {
		long sum = 0;
		for (Long long1 : values) {
			sum += long1;
		}

		return sum;
	}

	private void printMap() {
		Set<Entry<String, List<Long>>> entrySet = actionsTook.entrySet();
		System.out.println("ActionType;Min;Max;Average;Median;TotalSum;NumberOfValues");
		for (Entry<String, List<Long>> entry : entrySet) {
			System.out.println(entry.getKey() + "," + getMin(entry.getValue()) + "," + getMax(entry.getValue()) + ","
					+ getAverage(entry.getValue()) + "," + get-
Median(entry.getValue()) + "," + getSum(entry.getValue())
					+ "," + entry.getValue().size());

		}
	}

	private void parseFoundMatchTimes() {
		boolean isScanning = false;
		String marketIdScanned = "";
		LocalDateTime startTime = null;
		long actionTookMs = 0;
		List<Long> tookMs = Lists.newArrayList();
		for (int i = 0; i < fileAsLogLine.size(); i++) {

			LogLine logLine = fileAsLogLine.get(i);
			if (!isScanning && logLine.getActionType() == LogActionType.FOUND_MATCH_START) {
				isScanning = true;
				startTime = logLine.getD();
				marketIdScanned = logLine.getMarketId();
			}
			if (isScanning && logLine.getMarketId().equals(marketIdScanned)
					&& logLine.getActionType() == LogAction-
Type.LOCK_MARKET_START) {
```

```
                                                                                actionTookMs = logLine.getD().toInstant(ZoneOffset.UTC).toEpochMilli()
                                                                                                - startTime.toIn-
stant(ZoneOffset.UTC).toEpochMilli();
                                                                                isScanning = false;
                                                                                tookMs.add(actionTookMs);
                                                        }
                                        }
                                        actionsTook.put("FoundMatch", tookMs);
                        }

                        public static void main(String[] args) {
                                        LogAnalyzer la = new LogAnalyzer();
                                        la.analyze();
                        }

                        private void parseStartEndKeyToMap(LogActionType start, LogActionType end, String keyName) {
                                        boolean isScanning = false;
                                        String marketIdScanned = "";
                                        LocalDateTime startTime = null;
                                        long actionTookMs = 0;
                                        String threadName = "";

                                        List<Long> tookMs = Lists.newArrayList();
                                        for (int i = 0; i < fileAsLogLine.size(); i++) {

                                                        LogLine logLine = fileAsLogLine.get(i);
                                                        if (!isScanning && logLine.getActionType() == start) {
                                                                        isScanning = true;
                                                                        startTime = logLine.getD();
                                                                        marketIdScanned = logLine.getMarketId();
                                                                        threadName = logLine.getThreadName();
                                                        }
                                                        if (isScanning && logLine.getMarketId().equals(marketIdScanned) && logLine.getActionType() == end
                                                                                        && logLine.getThreadName().equals(thread-
Name)) {

                                                                        actionTookMs = logLine.getD().toInstant(ZoneOffset.UTC).toEpochMilli()
                                                                                        - startTime.toIn-
stant(ZoneOffset.UTC).toEpochMilli();
                                                                        isScanning = false;
                                                                        tookMs.add(actionTookMs);
                                                        }
                                        }
                                        actionsTook.put(keyName, tookMs);
                        }

                        private static LogActionType getLogActionType(String mesg) {
                                        if (mesg.contains("found matched bet with betid")) {
                                                        return LogActionType.FOUND_MATCH_START;
                                        }
                                        if (mesg.contains("started locking market")) {
                                                        return LogActionType.LOCK_MARKET_START;
                                        }
                                        if (mesg.contains("locking market done")) {
                                                        return LogActionType.LOCK_MARKET_END;
                                        }
                                        if (mesg.contains("started getMarket from cache.")) {
                                                        return LogActionType.GET_MARKET_FROM_CACHE_START;
                                        }
                                        if (mesg.contains("getMarket from cache done")) {
                                                        return LogActionType.GET_MARKET_FROM_CACHE_END;
                                        }
                                        if (mesg.contains("updateProbabilities started.")) {
                                                        return LogActionType.UPDATE_PROBABILITIES_START;
                                        }
                                        if (mesg.contains("updateProbabilities done")) {
                                                        return LogActionType.UPDATE_PROBABILITIES_END;
                                        }
                                        if (mesg.contains("updateProbabilities() deserialized from Redis started")) {
                                                        return LogActionType.UPDATE_PROBABILITIES_DESERIALIZE_START;
                                        }
```

```
if (mesg.contains("updateProbabilities() deserialized from Redis done")) {
                return LogActionType.UPDATE_PROBABILITIES_DESERIALIZE_END;
}
if (mesg.contains("clearAmountHandled()1 started")) {
                return LogActionType.CLEAR_AMOUNT_1_START;
}
if (mesg.contains("clearAmountHandled()1 done")) {
                return LogActionType.CLEAR_AMOUNT_1_END;
}

if (mesg.contains("cancelRiskBets() started")) {
                return LogActionType.CANCEL_RISK_BETS_START;
}
if (mesg.contains("cancelRiskBets() finished")) {
                return LogActionType.CANCEL_RISK_BETS_END;
}

if (mesg.contains("clearAmountHandled()2 started")) {
                return LogActionType.CLEAR_AMOUNT_2_START;
}
if (mesg.contains("clearAmountHandled()2 done")) {
                return LogActionType.CLEAR_AMOUNT_2_END;
}

if (mesg.contains("cleanActiveBets() started")) {
                return LogActionType.CLEAN_ACTIVE_BETS_START;
}
if (mesg.contains("cleanActiveBets() done")) {
                return LogActionType.CLEAN_ACTIVE_BETS_END;
}

if (mesg.contains("getAvailableOffersExcludingOwnBets() started")) {
                return LogActionType.GET_AVAILABLE_OFFERS_START;
}
if (mesg.contains("getAvailableOffersExcludingOwnBets() done")) {
                return LogActionType.GET_AVAILABLE_OFFERS_END;
}
if (mesg.contains("createBetsAndCancellations() started.")) {
                return LogActionType.CREATE_BETS_START;
}
if (mesg.contains("createBetsAndCancellations() done")) {
                return LogActionType.CREATE_BETS_END;
}

if (mesg.contains("truncateBetOnMaxAllowedLiability() started")) {
                return LogActionType.TRUNCATE_BET_START;
}
if (mesg.contains("truncateBetOnMaxAllowedLiability() finished")) {
                return LogActionType.TRUNCATE_BET_END;
}

if (mesg.contains("performBetCancellationsAndBetPlacements() started")) {
                return LogActionType.PERFORM_BET_CANCELLATION_START;
}
if (mesg.contains("performBetCancellationsAndBetPlacements() finished")) {
                return LogActionType.PERFORM_BET_CANCELLATION_END;
}

if (mesg.contains("storeReporting started")) {
                return LogActionType.STORE_REPORTING_START;
}
if (mesg.contains("storeReporting done")) {
                return LogActionType.STORE_REPORTING_END;
}
if (mesg.contains("saveORMarket() started")) {
                return LogActionType.SAVE_ORMARKET_START;
}
if (mesg.contains("saveORMarket() done")) {
                return LogActionType.SAVE_ORMARKET_END;
}
return null;
```

```
	}

	private static class LogLine {
		LocalDateTime d;
		String threadName;
		String marketId;
		String message;
		LogActionType actionType;

		public LogLine(LocalDateTime d, String threadName, String marketId, String message, LogActionType actionType) {
			super();
			this.d = d;
			this.threadName = threadName;
			this.marketId = marketId;
			this.message = message;
			this.actionType = actionType;

		}

		public LocalDateTime getD() {
			return d;
		}

		public String getThreadName() {
			return threadName;
		}

		public String getMarketId() {
			return marketId;
		}

		public String getMessage() {
			return message;
		}

		public LogActionType getActionType() {
			return actionType;
		}

		@Override
		public String toString() {
			return "LogLine [d=" + d + ", threadName=" + threadName + ", marketId=" + marketId + ", message=" +
message
					+ ", actionType=" + actionType + "]";
		}

	}
}
```

Liite 3. LogAnalyzer.java luokan yksikkötestit

```
package com.joppe;

import static org.junit.Assert.assertNotEquals;
import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.List;
import java.util.Map;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class LogAnalyzerTest {
	LogAnalyzer logAnalyzer;

	@BeforeEach
	void setUp() throws Exception {
		logAnalyzer = new LogAnalyzer();
		// Create sample data where we have one line of start and end for each gathered
		// action
		String line1 = "2018-10-10 07:37:35.820 [pool-2-thread-4] - MarketId 1.149317007 started locking market.";
		String line2 = "2018-10-10 07:37:35.824 [pool-2-thread-4] - MarketId 1.149317007 locking market done.";
		String line3 = "2018-10-10 10:19:30.925 [pool-2-thread-25] - MarketId 1.149317007 getAvailableOffersExcludingOwnBets() started.";
		String line4 = "2018-10-10 10:19:31.033 [pool-2-thread-25] - MarketId 1.149317007 getAvailableOffersExcludingOwnBets() done.";
		String line5 = "2018-10-10 10:19:31.033 [pool-2-thread-25] - MarketId 1.149317007 createBetsAndCancellations() started.";
		String line6 = "2018-10-10 10:19:31.034 [pool-2-thread-25] - MarketId 1.149317007 createBetsAndCancellations() done.";
		String line7 = "2018-10-10 10:19:31.033 [pool-2-thread-25] - MarketId 1.149317007 truncateBetOnMaxAllowedLiability() started.";
		String line8 = "2018-10-10 10:19:31.033 [pool-2-thread-25] - MarketId 1.149317007 truncateBetOnMaxAllowedLiability() finished.";
		String line9 = "2018-10-10 10:19:30.925 [pool-2-thread-25] - MarketId 1.149317007 cleanActiveBets() started.";
		String line10 = "2018-10-10 10:19:30.925 [pool-2-thread-25] - MarketId 1.149317007 cleanActiveBets() done.";
		String line11 = "2018-10-10 10:19:03.054 [pool-2-thread-27] - MarketId 1.149276330 cancelRiskBets() started.";
		String line12 = "2018-10-10 10:19:03.054 [pool-2-thread-27] - MarketId 1.149276330 cancelRiskBets() finished.";
		String line13 = "2018-10-10 10:19:03.053 [pool-2-thread-27] - MarketId 1.149276330 clearAmountHandled()1 started.";
		String line14 = "2018-10-10 10:19:03.054 [pool-2-thread-27] - MarketId 1.149276330 clearAmountHandled()1 done.";
		String line15 = "2018-10-10 10:19:04.748 [pool-2-thread-3] - MarketId 1.149317007 started getMarket from cache.";
		String line16 = "2018-10-10 10:19:04.752 [pool-2-thread-3] - MarketId 1.149317007 getMarket from cache done.";
		String line17 = "2018-10-10 10:19:01.199 [pool-2-thread-8] - MarketId 1.149317007 performBetCancellationsAndBetPlacements()
started.";
		String line18 = "2018-10-10 10:19:01.591 [pool-2-thread-8] - MarketId 1.149317007 performBetCancellationsAndBetPlacements() fin-
ished.";
		String line19 = "2018-10-10 10:19:01.169 [pool-2-thread-30] - MarketId 1.149317007 saveORMarket() started.";
		String line20 = "2018-10-10 10:19:01.171 [pool-2-thread-30] - MarketId 1.149317007 saveORMarket() done.";
		String line21 = "2018-10-10 10:19:01.591 [pool-2-thread-8] - MarketId 1.149317007 storeReporting started.";
		String line22 = "2018-10-10 10:19:01.592 [pool-2-thread-8] - MarketId 1.149317007 storeReporting done.";
		String line23 = "2018-10-10 10:19:05.521 [pool-2-thread-3] - MarketId 1.149317007 truncateBetOnMaxAllowedLiability() started.";
		String line24 = "2018-10-10 10:19:05.521 [pool-2-thread-3] - MarketId 1.149317007 truncateBetOnMaxAllowedLiability() finished.";
		String line25 = "2018-10-10 10:19:05.911 [pool-2-thread-43] - MarketId 1.149317007 updateProbabilities started.";
		String line26 = "2018-10-10 10:19:05.911 [pool-2-thread-43] - MarketId 1.149317007 updateProbabilities() deserialized from Redis
started.";
		String line27 = "2018-10-10 10:19:05.912 [pool-2-thread-43] - MarketId 1.149317007 updateProbabilities() deserialized from Redis
done.";
		String line28 = "2018-10-10 10:19:05.912 [pool-2-thread-43] - MarketId 1.149317007 updateProbabilities done.";
		logAnalyzer.processLine(line1);
		logAnalyzer.processLine(line2);
		logAnalyzer.processLine(line3);
		logAnalyzer.processLine(line4);
		logAnalyzer.processLine(line5);
		logAnalyzer.processLine(line6);
		logAnalyzer.processLine(line7);
		logAnalyzer.processLine(line8);
		logAnalyzer.processLine(line9);
		logAnalyzer.processLine(line10);
		logAnalyzer.processLine(line11);
		logAnalyzer.processLine(line12);
		logAnalyzer.processLine(line13);
		logAnalyzer.processLine(line14);
		logAnalyzer.processLine(line15);
		logAnalyzer.processLine(line16);
```

```
                logAnalyzer.processLine(line17);
                logAnalyzer.processLine(line18);
                logAnalyzer.processLine(line19);
                logAnalyzer.processLine(line20);
                logAnalyzer.processLine(line21);
                logAnalyzer.processLine(line22);
                logAnalyzer.processLine(line23);
                logAnalyzer.processLine(line24);
                logAnalyzer.processLine(line25);
                logAnalyzer.processLine(line26);
                logAnalyzer.processLine(line27);
                logAnalyzer.processLine(line28);

        }

        @AfterEach
        void tearDown() throws Exception {
        }

        @Test
        void testProcessLine() {
                assertEquals(28, logAnalyzer.getFileAsLogLineSize());

        }

        @Test
        void testAnalyze() {
                logAnalyzer.analyze();
                Map<String, List<Long>> actionsTook = logAnalyzer.getActionsTook();
                // Test that output isn't empty dataset
                assertNotEquals(0, actionsTook.size());

                // Test that each milliseconds are calculated correct from the sample set
                List<Long> list = actionsTook.get("lockMarket");
                assertEquals((Long) 4L, (Long) list.get(0));
                list = actionsTook.get("createBets");
                assertEquals((Long) 1L, (Long) list.get(0));
                list = actionsTook.get("getAvailableOffers");
                assertEquals((Long) 108L, (Long) list.get(0));
                list = actionsTook.get("truncateBet");
                assertEquals((Long) 0L, (Long) list.get(0));

                list = actionsTook.get("cancelRiskBets");
                assertEquals((Long) 0L, (Long) list.get(0));
                list = actionsTook.get("cleanActiveBets");
                assertEquals((Long) 0L, (Long) list.get(0));

                list = actionsTook.get("clearAmount1");
                assertEquals((Long) 1L, (Long) list.get(0));

                list = actionsTook.get("getMarketFromCache");
                assertEquals((Long) 4L, (Long) list.get(0));

                list = actionsTook.get("performBetCancellation");
                assertEquals((Long) 392L, (Long) list.get(0));

                list = actionsTook.get("saveOrMarket");
                assertEquals((Long) 2L, (Long) list.get(0));

                list = actionsTook.get("storeReporting");
                assertEquals((Long) 1L, (Long) list.get(0));
                list = actionsTook.get("truncateBet");
                assertEquals((Long) 0L, (Long) list.get(0));
                list = actionsTook.get("updateProbDeserialize");
                assertEquals((Long) 1L, (Long) list.get(0));
                list = actionsTook.get("updateProbabilities");
                assertEquals((Long) 1L, (Long) list.get(0));

        }

}
```