Tieteellisestä laskennasta Helsingin yliopiston Kumpulan kampuksella

Frans Ojala

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 8. toukokuuta 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Frans Ojala

Työn nimi — Arbetets titel — Title
Tieteellisestä laskennasta Helsingin yliopiston Kumpulan kampuksella

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	8. toukokuuta 2015	21

Tiivistelmä — Referat — Abstract

Hajautettu rinnakkaislaskenta on noussut viime vuosina yhdeksi tärkeimmistä tieteellisen laskennan välineistä. Myös teollisuus on ottanut rinnakkaislaskennan laajasti käyttöön. Tutkielmassa tarkastellaan tämänhetkisiä hajautetun laskennan ohjelmistoarkkitehtuureita ja sovellutuksia. Esittelyssä on myös Helsingin yliopiston tietojenkäsittelytieteen laitoksen laskentaklusteri Ukko. Ukko-klusterin käyttöasteesta kerättiin tietoja tammi- huhtikuun aikana, tärkeimmät havainnot esitellään. Tutkielmaa varten haastateltiin henkilöitä Helsingin yliopiston Kumpulan kampukselta. Haastatteluiden ja Ukko-klusterista kerättyjen tietojen perusteella oli mahdollisista päätellä parannuksia Ukko-klusterille.

ACM Computing Classification System (CCS):
- Computer systems organization → Distributed architectures
- Computer systems organization → Grid computing
- Networks → Cloud computing
- Computing methodologies → Parallel computing methodologies
- Computing methodologies → Distributed computing methodologies
- Applied computing
- Software and its engineering → Distributed programming languages

Avainsanat — Nyckelord — Keywords
HPC, hajautetut järjestelmät, käyttöaste, tieteellinen laskenta, hajautettu laskenta, MapReduce

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto **1**

2 Määritelmiä **2**

3 Laskentaklusteri Ukko **3**
3.1 Käyttö keväällä 2015 4
3.2 Havaintoja 5

4 Hajautetusta laskennasta **8**
4.1 Hajautetun laskennan ohjelmistoista 8
4.1.1 MapReduce 8
4.1.2 Apache Hadoop 9
4.1.3 Apache Spark 9
4.1.4 MPI 10
4.1.5 Hama 11
4.2 Tietovarannoista 11
4.2.1 HDFS 11
4.2.2 Cassandra 12
4.2.3 Lustre 12
4.3 Kokonaisvaltaisista ratkaisuista 13
4.3.1 OpenStack 13
4.3.2 Apache Mesos 14
4.3.3 EUCALYPTUS 15

5 Tieteellinen laskenta Kumpulan kampuksella **15**

6 Johtopäätöksiä **16**

Lähteet **17**

1 Johdanto

Tieteenalojen kehittyessä etenkin matemaattis-luonnontieteellisillä aloilla tieteellisestä laskennasta on muodostunut yksi tärkeimmistä työkaluista ongelmanratkaisussa. Esimerkiksi uusien teoriasuuntausten validoinnissa käytetään simulaatioita runsaasti. Vuosien saatossa tarve laskentakapasiteetille on vain kasvanut ja kasvaa edelleen kun tarvitaan tarkempia simulaatioita tai louhitaan suurempia datamassoja. Tarpeita tyydyttämään on kehitetty toinen toistaan tehokkaampia tietokonejärjestelmiä [Nas90]. Tietomäärien kasvaessa on havaittu, että perinteiset supertietokoneet joutuvat verraten vaatimattoman suoritustehon takia väistymään uuden sukupolven tietojärjestelmien tieltä [Kos06].

Modernit massiiviset laskentajärjestelmät voidaan kategorisoida kolmeen luokkaan: klusterit, gridit ja pilvet. Klusterit ovat tyypillisesti yhden organisaation tai instituution omistamia järjestelmiä jotka sijaitsevat heidän tiloissaan. Gridit ovat monesti maiden rajat ylittäviä järjestelmiä jotka voivat koostua mm. erillisistä klustereista. Laskentapilvet ottavat hieman erilaisen lähestymistavan. Pilvet ovat rakennettu tarjoamaan laskentakapasiteettia silloin kun sitä tarvitaan niin, että palveluntarjoajan ja asiakkaan välinen palveluehtosopimus täyttyy [SK11]. Pohjimmiltaan kaikki ratkaisut on kehitetty samankaltaista tarkoitusperää varten: tarjoamaan massiivista laskentakapasiteettia [FZRL08].

Helsingin Yliopiston tietojenkäsittelytieteen laitoksen laskentajärjestelmä on edellisessä erittelyssä tyypiltään klusteri. Sen vapaat solmut ovat laitoksen opiskelijoiden ja henkilökunnan käytössä kellon ympäri ja niitä voi varata erityiskäyttöön[1]. Pyynnöstä myös muut tiedekuntalaiset pääsevät käyttämään klusterin tarjoamaa laskentakapasiteettia. Myöhemmin esiteltävästä datasta selviää, että *Ukko*-klusteria käytetään varsin poikkitieteellisesti esimerkiksi bioinformatiikan ja fysiikan laskennan sovellutuksissa tietojenkäsittelytieteen ongelmanratkaisun lisäksi.

Ukko-klusterin käyttö on kuitenkin ollut varsin vähäistä. On syytä pohtia toisivatko uudet hallintaohjelmistot sekä hajautettuun laskentaan tarkoitetut sovelluskehykset käytön helppoutta ja käyttöasteen kasvua. Hallintaohjelmistot tarjoavat ratkaisuja suoritinkuorman, verkkoliikenteen, levy I/O:n ja monen muun muuttujan tarkkailuun ja optimointiin. Lisäksi monet hallintaohjelmistot tarjoavat ratkaisuja nimensä mukaisesti solmujen ja koko klusterin hallintaan: päivityksiin, ohjelmistojen asentamiseen, uudelleenkäynnistyksiin jne.. Sovelluskehykset puolestaan tarjoavat helppokäyttöisiä ohjelmointirajapintoja hajautettujen laskentaohjelmistojen toteutukseen [DG08]. Ohjelmointirajapinnat piilottavat hajautettujen solmujen ja -ohjelmien käyttöön liittyvät monimutkaisuudet sovelluskehyksen hoidettavaksi, jolloin oh-

[1] Laskentaklusteri Ukko: http://www.cs.helsinki.fi/tietotekniikka/laskentaklusteri-ukko (2015)

jelmoijat voivat keskittyä ongelman ratkaisuun [ZCD+12].

Tutkielmassa esitellään tämänhetkisiä suuntauksia hajautettujen laskentajärjestelmien alalta. Tiedot Ukko-klusterilla ajossa olleista ohjelmistoista on kerätty tammi- huhtikuussa 2015. Kyselyt joihin arviot klusterin käytöstä tulevaisuudessa myös perustuvat ovat toteutettu helmi- huhtikuussa 2015. Näiden tietojen avulla on pyritty kartoittamaan mahdollisuuksia nostaa Ukko-klusterin laskentakapasiteettia ja käyttöastetta.

2 Määritelmiä

Etenkin pilven määritelmä on liikkuva käsite mutta tässä tutkielmassa nojaudutaan lähinnä alla olevaan määrittelyyn. Annetaan edellä mainitulle kolmelle rinnakkaisten järjestelmien luokittelulle hieman tarkemmat määritelmät, jotta on mahdollista tarkastella missä erilaiset HPC-arkkitehtuurit ovat parhaimmillaan.

Klusteri on kokoelma rinnakkaisia hajautettuja tietokoneita, solmuja, jotka on kytketty toisiinsa nopealla tiedonsiirtoväylällä kuten Gigabit Ethernet tai Infiniband. Yksittäistä tietokonetta vasten etuna klustereissa on erityisesti massiivinen rinnakkaisuus ja solmujen varakopiot – yhden laskentasolmun rikkoutuessa laskenta ei jää kesken, vaan kyseinen osa laskennasta voidaan suorittaa toisella solmulla. Käyttäjän kannalta klusteri on edelleen kokoelma yksittäisiä tietokoneita mutta niitä käytetään kuten loogisesti yhteneväistä tietokonejärjestelmää. Massiivisen rinnakkaisuuden myötä laskenta pyritään jakamaan tasaisesti kaikille solmuille. Tällöin laskentakapasiteetti on teoriassa kaikkien solmujen yhteenlaskettu kapasiteetti. [SK11, HMH+13]

Gridi on edelleen kokoelma yhteen kytkettyjä rinnakkaisia tietokoneita. Pääasiallinen ero klusteriin on niiden hallintoalue. Klusterit ovat fyysisesti ja loogisesti yhden hallintoalueen piirissä, kun taas gridit ovat tyypillisesti usean hallintoalueen piirissä. Tyypillisesti gridit koostuvat erillisten instituutioiden ylläpitämistä klustereista jotka on loogisesti liitetty yhteen internetin välityksellä. Samoin kun klusterin solmut saattavat verkko-ongelmien tai rikkoutumisen takia poistua käytöstä äkillisesti, voi myös gridiin osallistuva laskentayksikkö kadota. Kun gridiympäristössä laskennan hajauttaminen toteutetaan julkisen internetin välityksellä, täytyy internetin epäluotettavuus yhteyskanavana ottaa huomattavasti enemmän huomioon. [SK11, HMH+13]

Pilven tarkoitus on tarjota asiakkaille laskentakapasiteettia ja ohjelmistopalveluita internetin välityksellä. Kapasiteetti ja palvelut ostetaan palveluntarjoajalta sitä mukaan kun niitä tarvitaan, näin pilvet tarjoavat nopean, luotettavan ja rahoituksellisesti kevyen tavan rakentaa ja saattaa markkinoille uusia ohjelmistoja sekä palveluita. Mikään yksittäinen arkkitehtuuri ei kuvaa kaikkia pilvien tarjoamia ominaisuuksia ja pilvellä on teollisuudessa ja tutkijayhteisöissä monta määritelmää. Pilvet tyypillisesti kuitenkin rakennetaan klusteri- ja gridityyppisistä ratkaisuista yhden hallinnollisen elimen,

kuten yrityksen, toimesta. Kuten gridien tapauksessa, organisaation levittäytyessä maailmanlaajuiseksi toimijaksi on pilven sisäisessä arkkitehtuurissa huomioitava solmujen ja kokonaisten laskentayksiköiden katoaminen ja/tai rikkoutuminen. Kansainvälisillä pilvitoimijoilla on lisäksi haasteenaan eri hallintoalueiden mahdollisesti paljonkin eriävät lait ja asetukset koskien mm. tietoliikennettä ja tietoturvaa. [Mat14, FZRL08]

3 Laskentaklusteri Ukko

Tietojenkäsittelytieteen laitoksen ylläpitämä laskentayksikkö, *Ukko*-klusteri, on nimensä mukaisesti klusteri. Se koostuu 240:stä Dell PowerEdge M610 -korttipalvelimesta, jotka on kytketty Gigabit Ethernet -yhteyksin toisiinsa, tiedostopalvelimeen sekä internetiin[2]. Kussakin solmussa on 32 Gt keskusmuistia ja niihin on asennettu Ubuntu Linux-käyttöjärjestelmä sekä tarvittavat etähallintaohjelmistot. Pääsy Ukon solmuille on rajattu laitoksen opiskelijoille ja henkilökunnalle ja tapahtuu vain yliopiston verkosta SecureShell-yhteyden ylitse. Ukkoa ei ole kytketty osaksi gridejä.

Kuten tavallista klusteriympäristössä, Ukkoa käytetään tyypillisesti yhdeltä solmulta käsin. Tällä hetkellä tyypillisin käyttötapaus on laskenta yhdellä solmulla, sillä klusterille ei ole asennettu rinnakkaislaskentaan tarkoitettuja sovelluksia tai sovelluskehyksiä. Käyttäjien ei ole mahdollista itsenäisesti asentaa ohjelmistoja Ukon solmuille, tämä on ylläpidon tehtävä. Toisaalta käyttäjien on mahdollista ajaa solmuilla mitä tahansa ohjelmistoja jotka eivät vaadi erillisiä asennuksia, eli ohjelmistoja joita voidaan käynnistää käyttäjän oikeuksin. Esimerkiksi *Spark*-rinnakkaislaskentaohjelmistoa on mahdollista käyttää 'standalone' tilassa, jolloin käyttäjä määrittelee sen eri komponenttien sijainnin ja käynnistää ne käsin[3]. 'Standalone'-tilassa vain kyseisellä käyttäjällä on pääsy hänen pystyttämäänsä Spark-laskentaympäristöön. Toisen käyttäjän pystyttäessä oman laskentaympäristönsä syntyy päällekkäisyyksiä jotka kuluttavat ylimääräisiä resursseja.

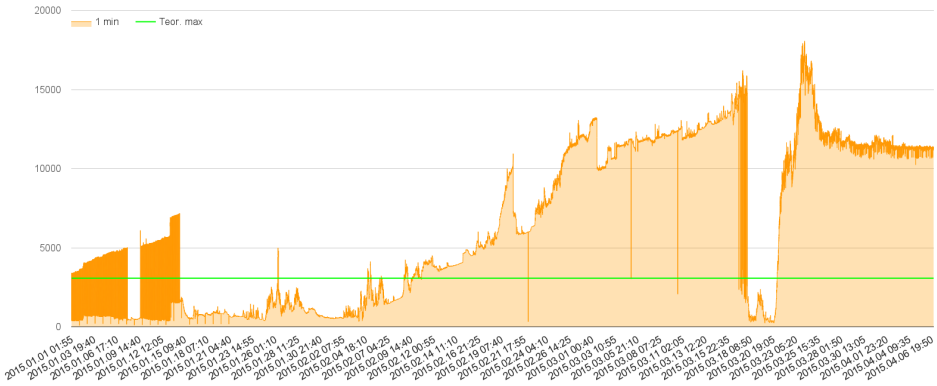
Ohjelmistojen asentamisen rajaaminen ylläpidolle on ymmärrettävästi tietoturvakysymys: monen käyttäjän ympäristössä olisi mahdollista, että kokematon käyttäjä asentaa tai konfiguroi ohjelmistoja sallien esimerkiksi järjestelmän käytön ulkopuolisille. Näin on kuitenkin mahdollista pitää järjestelmä yhtenäisenä, jotta sen ylläpito ei käy vaivalloiseksi. Kääntöpuolena käyttäjät ja tutkijat eivät voi asentaa ohjelmistoja jotka olisivat hyödyllisiä sekä heille, että muulle klusteria käyttävälle yhteisölle. Tutkijan varatessa solmun omaan käyttöönsä hän luonnollisesti saa täydet oikeudet käyttää kyseistä solmua.

[2]Laskentaklusteri Ukko: http://www.cs.helsinki.fi/tietotekniikka/laskentaklusteri-ukko (2015)

[3]Apache Spark Standalone Mode: https://spark.apache.org/docs/latest/spark-standalone.html (2015)

Hajautettujen laskentaohjelmistojen pystyttäminen ja ylläpito on resurssi-intensiivistä. Siksi tällä hetkellä tutkimusryhmien on helpompaa ja nopeampaa varata solmuja yksinomaan heidän käyttöön ja käyttää niitä ylläpitäjinä häiritsemättä muuta klusterilla tapahtuvaa toimintaa. Vaikuttaisi myös siltä, että erilaisten ongelmien ratkaisuun tarvitaan useampia laskentaohjelmistoja ja yksikään ohjelmisto ei voi tarjota kaikkea toiminnallisuutta yksinään [HKZ+11]. Kun samassa laskentaympäristössä on useampia ohjelmistokehyksiä kilpailemassa resursseista käy kokonaisuuden hallitseminen vieläkin vaikeammaksi ja sitä varten on keksittävä ratkaisuita [ZBSS+10]. Laajan ohjelmistovalikoiman pystyttäminen ja ylläpito vaatisi keskitettyä hallintaa ja sen helpottamiseksi onkin kehitetty ohjelmistoja, kuten *Apache Mesos*[4] ja *Mesosphere*[5].

3.1 Käyttö keväällä 2015



Kuva 1: Kokonaiskuorma klusterilla

Jotta oli mahdollista muodostaa kuvan Ukon käytöstä ja käyttöasteesta klusterilta kerättiin tietoja kevään 2015 aikana. Klusterilta kerätään jatkuvasti tietoa sen käyttöasteesta, mutta Tammikuussa 2015 tiedonkeruun suorittavia ohjelmia päivitettiin ja ne sisältävät nyt:

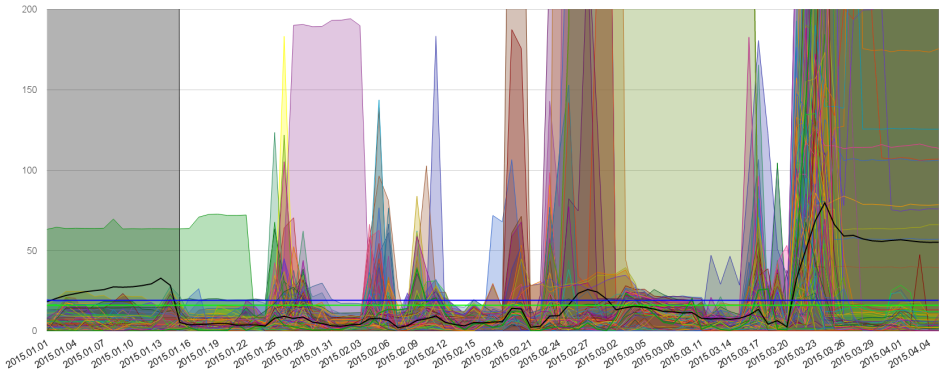
- prosessilistauksen (ps aux)
- vapaan muistin määrät (free)
- keskimääräisen kuormituksen (/proc/loadavg)
- levypalvelin I/O:n (nfsstat)
- verkkoliikennekuorman (/proc/net/dev)

[4]Apache Mesos: http://mesos.apache.org (2015)
[5]Mesosphere: http://mesosphere.com (2015)

- käyttöjärjestelmäversion (uname -a)
- sisäänkirjautuneet käyttäjät
- ohjelmistopäivitykset
- palvelimen raudan ominaisuudet (dmidecode)

Tiedot kerättiin kaikilta käytössä olevilta palvelimilta viiden minuutin välein. Palvelimet 193-240 olivat poistettu yleisestä käytöstä muuhun toimintaan eikä niiltä siksi kerätty tietoja. Aiemmin kerättyä dataa ei käytetty tutkielmassa, sillä se ei sisältänyt prosessilistauksia jotka ovat hyvin tärkeitä johtopäätösten tekemiseksi. Tietojen analyysi tehtiin yksinomaan Shell-skripteillä[6].

3.2 Havaintoja



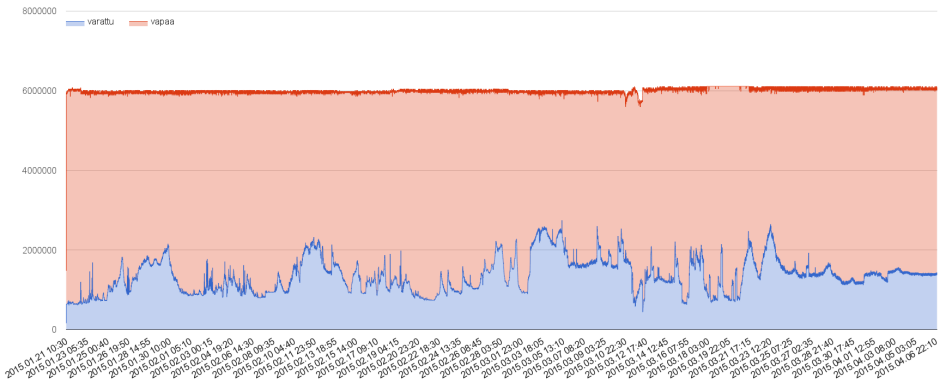
Kuva 2: Keskimääräinen kuorma solmuilla, päivän keskiarvo

Suoritinkuorma tallennettiin Ukko-klusterin solmuille asennetun Ubuntu-Linux käyttöjärjestelmän tiedostosta /proc/loadavg. Kuvaajaan 1 on kerätty kumulatiivinen suoritinkuorma koko klusterilta, vihreä viiva edustaa teoreettista maksimaalista hyötykuormaa[7]. Muutamia katkoja lukuun ottamatta kuvasta 1 selviää, että Ukko-klusterin kuormitus oli noussut huomattavasti

[6]R-ohjelmointikieltä käytetään monesti tilastollisessa analyysissä. Tässä sen käyttö ei kuitenkaan ollut mahdollista, sillä tiedot piti hajauttaa monelle Ukko-klusterin solmulle jotta niiden käsittely oli mahdollista keskusmuistin ja ajan puitteissa. Muita ohjelmistoja ei käytetty, sillä niitä ei ollut saatavilla. Tästä johtuen analyysi on puutteellinen ja vaatisi vahvempien työkalujen käyttöä sekä huomattavasti enemmän aikaa.

[7]Teoreettisella maksimaalisella hyötykuormalla viitataan tässä siihen kuormitukseen, jolla suorittimen ydin on optimaalisesti kuormitettu. Tällöin ytimellä suoritetaan hyödyllistä laskentaa mahdollisimman paljon suhteessa kontekstinvaihdoksiin. Ukon solmuilla on kaksi neliytimistä HyperThreading-suoritinta, tällöin teoreettinen maksimikuorma on noin 2*4*2 = 16. Lisää esimerkiksi osoitteessa: http://linux.die.net/man/5/proc kohta loadavg.

helmikuun puolen välin jälkeen. Kuvaajassa 2 on esitetty kuormitus yksittäisille solmuille, jossa musta käyrä on päivän keskiarvo koko klusterin ylitse. Sininen viiva on keskikuormitus koko tarkastelujaksolla, vihreä viiva taas teoreettinen maksimaalinen hyötykuorma. Kun tarkastellaan solmujen kuormitusta yksittäin, havaitaan, että kuormituksen kasvulle on pääasiallisena syynä muutamat hyvin kuormittuneet solmut.



Kuva 3: Vapaan muistin määrä koko klusterilla

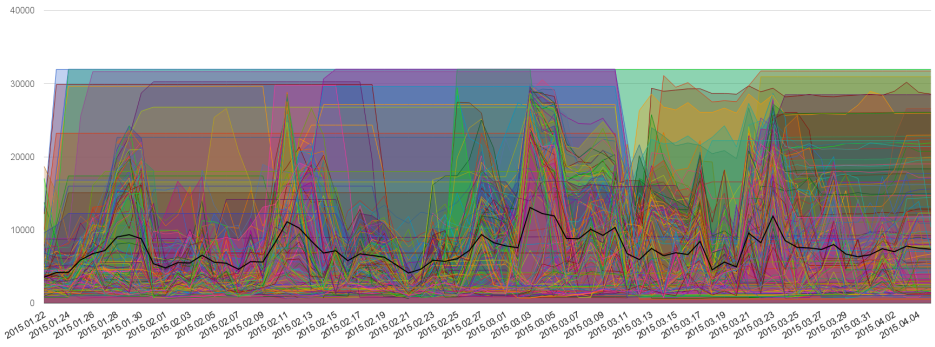
Yksittäisten solmujen kuormia analysoitaessa havaittiin, että solmulla 10 käynnistyi helmikuun alussa prosessi joka monistui nopeasti. Prosessi kasvatti solmun suoritinkuormaa lineaarisesti yli 12000, jonka jälkeen solmu kaatui maaliskuun puolessa välissä. Kyseisen 'fork-bomb'[8] prosessin takia sen kuormitusta ei ole kuvaajassa 2. Se on kuitenkin mukana klusterin yli tehdyssä keskiarvostuksessa kuvaajassa 1. Kuvaajien eroavaisuudesta on tärkeä huomioida, että kuormitusta on syytä tarkastella sekä koko klusterin ylitse että solmuittain. Tämä pätee erityisesti tilanteissa, joissa klusteria ohjelmoidaan pääasiassa kuin yksittäisiä tietokoneita, kuten Ukko-klusterilla nyt tehdään.

Kuvaajasta 2 selviää, että Ukko-klusterin solmujen kuormitus on tarkastelujaksolla keskimäärin teoreettisen maksimaalisen hyötykuorman paikkeilla. Huomionarvoista on kuitenkin, että 65 % ajasta kuormitus on paljon pienempi kuin teoreettinen maksimi. Tarkastelujakson aikana on selviä piikkejä joidenkin solmujen kuormituksessa, erityisesti alkaen maaliskuun lopusta. Pääasiassa solmujen kuormitus on siis ollut vähäistä ja jotkut solmut ovat ylikuormitettuja. Taulukkoon 1 on kerätty tietoja ajossa olleista prosesseista edellä mainittuissa piikeissä.

Muistinkäyttö klusterilla kerättiin käyttöjärjestelmän *free*-komennolla[9].

[8] Nimitys tulee ohjelmoinnissa käytetystä 'fork'-alirutiinista jolla isäntäprosessi voi luoda lapsiprosessin. Kun 'fork-bomb' käynnistyy alkaa isäntäprosessi luomaan lapsiprosesseja hallitsemattomasti ohjelmointivirheen takia.

[9] *Free*-komennon tuottamasta tulosteesta oleellista on puskureiden ja välimuistien koko



Kuva 4: Käytetyn muistin määrä solmuilla, päivän keskiarvo

Kuvaajaan 3 on kerätty kumulatiivinen muistinkäyttö koko klusterilta. Tarkasteluaikavälillä klusterissa oli muistia 32 Gt*192 solmua = 6144 Gt. Tarkasteltaessa käytettyä muistia havaitaan, että koko tarkastelujakson aikana muistista on käytössä vain murto-osa. Kuvaajassa 4 näkyy muistin käyttö solmuittain. Kuvaajasta selviää, että joillakin solmuilla muisti on kokonaan käytössä useita päiviä. Kuten suoritinkuormassa, on muistin käytössä selviä piikkejä joidenkin solmujen kohdalla.

Päivä	Käyttäjä	Solmuja	Ohjelmisto
25.1.	Käyttäjä 1	92	R-pohjainen
3.2.	Käyttäjä 2	1	Java-pohjainen
5.2.	Käyttäjä 3	15	Python-pohjainen
9.2.	Käyttäjä 4	57	Java-pohjainen
2.3.	Käyttäjä 5	93	Java-pohjainen
4.3.	Käyttäjä 4	5	Java-pohjainen
23.3.	Käyttäjä 4	105	Java-pohjainen

Taulukko 1: Kuormituspiikit

Samoin kuin suorittimen ylikuorma ei muistin ylikuorma ole toivottavaa pitkällä aikavälillä. Sadan prosentin muistinkäytössä solmu joutuu käyttämään runsaasti aikaa muistin sivutukseen ja *thrashing*-ilmiö voimistuu alentaen solmun laskentakapasiteettia huomattavasti. Tarkastelujakson aikana syy liialliselle suoritinkuormalle ja muistinkäytölle olivat käyttäjien prosessit.

ynnä vapaa muisti, joka kertoo kuinka paljon ajossa olevat ohjelmistot itse vievät muistia. Lisätietoja: http://linux.die.net/man/5/proc kohta meminfo.

4 Hajautetusta laskennasta

Hajautettua laskentakapasiteettia voi käyttää monen eri ongelman ratkaisussa. Toisaalta kukin ongelma-alue tarvitsee omanlaisensa tavan käyttää laskentakapasiteettia. Siinä missä esimerkiksi MapReduce ja PageRank pystyvät hyödyntämään valtavaa rinnakkaisuutta [DG08, KCN06] on olemassa ongelmia, joille ei tunneta ratkaisuja jotka hyötyisivät rinnakkaisuudesta. Nimellisesti, P-täydelliset ongelmat eivät hyödy rinnakkaisuudesta juurikaan ja niille pyritäänkin löytämään vaihtoehtoisia rinnakkaistettavia ratkaisuja [GHR95]. Vaikka algoritmillinen kompleksisuus vaikuttaa paljon järjestelmien vaatimuksiin, siihen ei keskitytä tässä tutkielmassa vaan syvennytään rinnakkaislaskentaan.

4.1 Hajautetun laskennan ohjelmistoista

Tässä kappaleessa tarkastellaan ohjelmistoja jotka tarjoavat menetelmiä massiivisen rinnakkaisuuden hyödyntämiseen. Sellaiset ohjelmointiympäristöt kuten *MapReduce*, *Apache Spark* ja *MPI* helpottavat rinnakkaisuuden hallintaa tarjoamalla rajapintoja joiden taakse on kätketty rinnakkaisten ohjelmien ajamiseen liittyvät monimutkaisuudet. Kaikkien kolmen rinnakkaislaskentaympäristön – klustereiden, gridien ja pilvien – tapauksessa järjestelmään kytkettyjä palvelimia ja niiden komponentteja rikkoutuu koko ajan. Vikatilanteisiin on siis varauduttava ja kohdeltava niitä normaalina rinnakkaislaskentaympäristön ominaisuutena poikkeustilanteen sijaan [GGL03].

Tarkasteluun on valittu pääasiassa avoimen lähdekoodin ohjelmistoja ja ohjelmistokehyksiä, sillä suljetun lähdekoodin vastineista on verraten vähän tietoa saatavilla. Kriteereinä ohjelmistokehyksen tai ohjelmiston valikoitumiseen käytettiin myös käytön suosiota ja kehityskaarta jotka kummatkin kielivät paremmuudesta verrokkien suhteen.

4.1.1 MapReduce

MapReduce -ohjelmointimalli kehitettiin *Google*lla vastaamaan heidän tarvettaan ohjelmointimallille joka olisi riittävän yksinkertainen mutta tehokas korvaamaan sadat erikoisohjelmistot suurien tietomäärien käsittelyyn. MapReduce saa nimensä funktionaalisen ohjelmointityylin kahdesta peruselementistä: *map* – joka muuntaa lähtödatan väliaikaisiin avain-arvo pareihin – ja *reduce* – joka muuntaa väliaikaiset avain-arvo parit lopulliseksi tulokseksi. MapReduce on riittävän tehokas jotta sillä voidaan ratkaista hyvin monta reaalimaailman ongelmaa kuten merkkijonojen etsiminen, käynnit tietyssä www-osoitteessa, käänteisen www-yhteysverkon luominen ja hajautettu järjestäminen. [DG08]

Toisaalta MapReduce toimii parhaiten ‘naurettavan rinnakkaistettaviin’ ongelmiin eikä tarjoa varteenotettavia menetelmiä esimerkiksi simulaatiopohjaiselle laskennalle [JSK12]. Erona perinteiseen funktionaaliseen ohjelmointiin

MapReduce on täysin hajautettu eikä ohjelmoijan tarvitse välittää hajautuksen monimutkaisuuksista. Varsinainen C++ toteutus kätkee taakseen mm. vikatilanteiden-, tiedon lokaliteetin-, tehtävien suorituksen- ja varmuuskopioiden hallinnan laskentaympäristössä. Käyttäjä voi halutessaan optimoida laskentaa koskevia muuttujia kuten tulosteiden reduce-operaatioita ja määritellä tulosten esitysmuotoa koskevia parametreja. [DG08]

4.1.2 Apache Hadoop

Hadoop sai alkunsa MapReducesta. Kuten sanottu, MapReduce kehitettiin *Google*lla ja on siten suljetun lähdekoodin ohjelmisto. Sen julkistuksen myötä *Nutch*-projektin kehitystiimi löysi *Google*:n MapReduce-käsitteistä ratkaisun kokemiinsa ongelmiin. Nutch-projektiin kehitetyt ominaisuudet eriytettiin Hadoop-projektiksi kun *Yahoo!* palkkasi toisen Hadoop-järjestelmän pääkehittäjistä, Doug Cutting:n, riveihinsä. Hieman myöhemmin Hadoop otettiin *Apache Software Foundation*in (myöh. AFS) korkean tason projektiksi. Hadoop on kasvanut sisältämään monta erillistä projektia ja on hyvin suosittu niin tutkijoiden kuin teollisuuden toimijoiden keskuudessa. [Whi09]

Hadoop järjestelmä käsittää hajautetun laskennan monta osa-aluetta, mutta sen pääasialliset komponentit ovat MapReduce-toteutus laskentaan ja Hadoop Distributed Filesystem (myöh. HDFS) tiedon massiiviseen tallentamiseen ja hakuun. [Whi09]

AFS-kehittäjät ja muu kehitysyhteisö on tehnyt paljon töitä Hadoop-järjestelmän kanssa. Sille on kehitetty suuri määrä lisäosia ja olemassa olevia osia on paranneltu. Hadoop on saanut kehitystä käytännössä kaikilla laskentaan liittyvillä osa-alueilla kuten töiden ajastus, tiedon liikkuminen, tallennus, kryptografia ja rinnakkaislaskenta grafiikkasuorittimilla. [PRGK14]

4.1.3 Apache Spark

Spark perustuu työhön jonka M. Zaharia ym. ovat tehneet Berkeleyn yliopistossa, Kaliforniassa. Työ käsitti hajautetun laskennan tärkeän osa-alueen johon mm. MapReduce-toteutuksissa ei ole otettu kantaa: välitulosten uudelleenkäyttäminen. Heidän työnsä tuloksena syntyi *Resilient Distributed Dataset* -hajautusparadigma (myöh. RDD) ja *Spark*-laskentaohjelmisto. [ZCD+12]

RDD-perustietotyyppi on esimerkiksi klusterin solmujen keskusmuistiin hajautettu kokoelma mitä tahansa dataa. Se rakentuu esimerkiksi HDFS:stä luetusta tekstitiedosta. RDD:lle voidaan tehdä esimerkiksi *map*, *reduce* ja *filter* tyyppisiä muunnoksia joiden tuloksena saatu RDD on mahdollisesti toista alatyyppiä. Kukin muunnos, eli laskenta-askel, tuottaa RDD:n. Jokainen RDD on osa pidempää laskentaketjua jossa RDD:n ei kuitenkaan tarvitse olla vielä laskettu, vaan se voidaan laskea lennossa juuri ennen kuin sen sisältämää dataa tarvitaan seuraavassa laskenta-askeleessa. RDD:n ehdottomasti tärkein ominaisuus on keskusmuistivetoinen laskenta, jolloin dataa

– joka on laskettu mahdollisesti monta laskenta-askelta sitten – ei tarvitse hakea levyltä tai kirjoittaa levylle jotta sitä voidaan käyttää tulevaisuudessa. Hitaiden levyoperaatioiden poisjäämisen myötä laskenta nopeutuu valtavasti. [ZCD+12]

Spark on RDD-hajautusparadigman toteuttava laskentaohjelmisto. Spark:lle on olemassa *Scala*, *Java* ja *Python* -kieliset ohjelmointirajapinnat. Spark-laskentaympäristön voi pystyttää 'standalone' tilaan mihin tahansa tietokoneille johon käyttäjällä on pääsy ja laskentaympäristöön voi konfiguroida useita solmuja. Tällöin käytetään Spark:n sisäänrakennettua klusterinhallintaa. Spark-laskentaympäristön voi myös pystyttää klusteriin tai pilveen esimerkiksi olemassa olevaan *Mesos* tai *Hadoop YARN* -ympäristöön, jolloin saman Spark-laskentaympäristön jakaminen useamman käyttäjän kanssa on mahdollista. [10]

4.1.4 MPI

MPI eli *Message Passing Interface* on laaja rajapintamäärittely rinnakkaislaskentaan tarkoitetuille ohjelmointikirjastoille. Sen kehitys aloitettiin 1992 ja sitä kehitetään edelleen. Rajapintamäärittely kattaa mm. miten kahden pisteen välinen kommunikointi tulee järjestää, tietotyyppien määrittelyn, kollektiivisen kommunikoinnin, ryhmien, kontekstien ja välimuistin hallinnan, prosessitopologiat sekä -luonnin ja -hallinnan, järjestelmäympäristön määrittelyn ja niin edelleen. Toisin sanoen lähes kaiken sen mitä perinteisiä, yhdellä tietokoneella suoritettavia, ohjelmistokirjastoja kirjoitettaessa tulee ottaa huomioon ja lisäksi paljon sellaista joka saa erityishuomiota hajautettuja ohjelmistoja kirjoitettaessa[11]. MPI ja sen toteuttavat kirjastot ovat kasvaneet erittäin merkittävään asemaan massiivisen rinnakkaisuuden laskenta-arkkitehtuureille ohjelmoitaessa; suurin osa laskentaohjelmistoista käyttää MPI-kirjastoja rinnakkaisuuden hallintaan [SKP06]. Vaikka rajapintamäärittely on erittäin kattava se ei sisällä esimerkiksi ohjelmistojen oikeanmukaisuuden tarkistukseen tai virheiden etsimiseen liittyviä määrittelyitä [GKS+11].

Jokaisen tutkijan ei kuitenkaan tarvitse kirjoittaa kirjastofunktioita uudestaan kun hän tarvitsee MPI-järjestelmän tarjoamaa rinnakkaisuuden hallintaa laskentaansa. Tätä varten on toteutettu useita sekä avoimen- että suljetun lähdekoodin kirjastoja. Ensimmäisen MPI-toteutuksen MPICH:n [GLDS96][12] jälkeen on julkaistu lukuisia muita: MVAPICH-kirjasto on todettu hyvin toteutetuksi sekä erittäin hyvin skaalautuvaksi Infiniband-verkkoympäristössä [SKP06]. Muita ovat mm. avoimen lähdekoodin Open MPI[13], Java-kielinen

[10] Apache Spark: https://spark.apache.org/docs/latest/index.html (2015)

[11] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 3.0: http://mpi-forum.org/docs/docs.html (2015)

[12] MPICH: http://www.mpich.org (2015)

[13] Open MPI: http://www.open-mpi.org (2015)

JMPI [Din99] sekä *Microsoft*:n .NET kielelle toteutettu MPI.NET[14].

MPI-arkkitehtuuria on käytetty erittäin laajalti monen eri tieteenalan laskentaintensiivisissä ongelmaratkaisuissa sen monimuotoisuuden vuoksi. Esimerkiksi [WMPX11]: Monte-Carlo simulointi, [FGS06]: merimallinnus, [ORI+10]: sähkömagneettiset simulaatiot. Toisaalta MPI-ohjelmien toteuttaja joutuu ottamaan kantaa lähes kaikkeen hajautetun ohjelman toimintaan, kuten vikatilanteista toipumiseen solmun rikkoutuessa [GL04], joka tekee kehityksestä hidasta verrattuna esimerkiksi Spark-järjestelmään.

4.1.5 Hama

Hama on yleinen BSP (Bulk Synchronous Parallel) ohjelmointitekniikoihin tarkoitettu ohjelmistokehys jota kehitetään korkean tason ASF-projektina[15]. BSP käsittää Haman tapauksessa matriisi sekä verkkoihin perustuvat massiiviset laskennat. Hama:n voi konfiguroida hyödyntämään Hadoop MapReduce -koneiston, Hama:an sisäänrakennetun hajautuskoneiston sekä *Microsoft Dryad*-järjestelmän tarjoamia rinnakkaisuuden hallinnan palveluita. Se on suunniteltu avaamaan massiivinen rinnakkaislaskenta numeraalisille menetelmille, tiedon louhinnalle, fysiikan mallinnukselle sekä erityisesti matriisi-inversioille. [SYK+10]

4.2 Tietovarannoista

Dataa tallennetaan tänä päivänä hurjaa vauhtia. Muun muassa erilaiset sosiaalisen median palvelut, kuten *Facebook* ja *Google+*, tallentavat käyttäjiä koskevaa dataa päivittäin jopa petatavuja [PRGK14]. Lisäksi etenkin 'Internet of Things' (myöh. IoT) -ilmiön myötä erityisesti erilaisista sensoreista kerätty data tulee kasvamaan entisestään [NOOY13]. Jotta laskentaa voidaan tehdä petatavujen skaaloissa, on data säilöttävä johonkin. Tera- ja petatavujen tietomäärien tallentaminen ja käyttö ei ole yksinkertaista, eikä se onnistu tehokkaasti perinteisillä palvelinratkaisuilla. Siksi on kehitetty hajautettuja tiedostojärjestelmiä jotka tarjoavat erilaisia ominaisuuksia erilaisen datan massiiviseen säilömiseen ja sen käyttöön [SLBA11]. Tarkastellaan muutamia – jälleen kerran avoimen lähdekoodin – ratkaisuja.

4.2.1 HDFS

HDFS on avoimen lähdekoodin Hadoop-järjestelmän tiedostojärjestelmäkomponentti. HDFS koostuu kahdesta komponentista: NameNode ja DataNode. NameNode sisältää tiedostojärjestelmään tallennettujen tiedostojen metadatan (eli DataNode-sijainnit), ja attribuutit kuten ext-tiedostojärjestelmissä.

[14]MPI.NET: http://www.osl.iu.edu/research/mpi.net/ (2015)

[15]Apache Hama: https://hama.apache.org (2015)

DataNode sisältää itse tiedoston kopion. HDFS voidaan konfiguroida hyvin vikasietoiseksi – se on tietoinen palvelinkehyksistä ja mm. sen tiedostojen kaksintamista voidaan räätälöidä. Toisin kuin esimerkiksi Lustre-tiedostojärjestelmässä vikasietoisuus HDFS:ssä saavutetaan pääasiassa edellä mainituilla ominaisuuksilla (nimellisesti tiedostojen kaksintamisella) RAID-mekanismien sijaan. [SKRC10]

HDFS on saavuttanut suuren suosion. Tämä näkyy sekä käyttöönottomäärissä, että organisaatioiden laadussa. HDFS:n ovat ottaneet käyttöön isot toimijat kuten *Adobe*, *Facebook*, *EBay* ja *Google*[16]. Suosio johtunee helposta käyttöönotosta, valtavasta skaalautuvuudesta, saatavuudesta avoimen lähdekoodin projektina sekä jatkuvasta kehityksestä [PRGK14].

4.2.2 Cassandra

Cassandra muistuttaa perinteistä relaatiotietokantaa. Se ei kuitenkaan tarjoa täyttä relaatiomallia tiedon tallennukseen vaan yksinkertaistetun tallennusmallin. Cassandra:ssa tieto tallennetaan moniulotteiseen tauluun, joka indeksoidaan avaimen perusteella. Arvo on rakennetta sisältävä objekti. Kuten Google:n BigTable-tietokannassa Cassandran taulun sarakkeita voidaan ryhmitellä ja näin tehostaa tiedonhakua. Cassandra suunniteltiin erittäin skaalautuvaksi toimimaan klustereissa ja pilvissä jotka on rakennettu hyllytavarana saatavista komponenteista. Erilaisten vikatilanteiden aiheuttamat poikkeamat tiedon yhtenäisyydessä on otettu siksi laajalti huomioon suunnittelussa ja toteutuksessa. Cassandran noudattamia yhteneväisyystakeita voidaan joustavasti säätää jotta erilaisia tietotyyppejä voidaan käsitellä asianmukaisella tehokkuudella. Pääasiallinen tekijä Cassandran skaalautuvuuteen on tiedon tehokas partitiointi solmujen ylitse. [LM10]

Cassandra on käytössä monissa suurissa kansainvälisissä yhtiöissä ja tyypillisestä avoimen lähdekoodin projektista poiketen Cassandralle on saatavilla palvelusopimuksia[17].

4.2.3 Lustre

Lustre rakentuu Linux käyttöjärjestelmien tukemalle Lustre-tiedostojärjestelmälle. Sen tarjoama tiedostojärjestelmärajapinta on täysin POSIX-yhteensopiva ja toimii pienin rajoituksin kuten paikallinen etxt4-tiedostojärjestelmä. Lustre on tällä hetkellä johtoasemassa tietovarantona HPC-arkkitehtuureissa, sillä sen tallennuskapasiteetti ja nopeus skaalautuvat tuhansien petatavujen kokoihin ja teratavujen nopeuksiin. [YVCJ07]

Lustre koostuu viidestä komponentista: Metadata Magement Service (MGS), Metadata Server (MDS), Metadata Target (MDT), Object Storage Server (OSS) ja Object Storage Target (ODT). MGS tallentaa Lustre

[16] Apache Hadoop, PoweredBy: http://wiki.apache.org/hadoop/PoweredBy (2015)
[17] Apache Cassandra: https://cassandra.apache.org (2015)

klusterin konfiguraatiotiedot, jakaa sitä asiakasohjelmistoille ja ottaa sitä vastaan muilta komponenteilta. MDS tarjoilee asiakasohjelmistoille MDT:n sisältämää metadataa ja hallinnoi verkkoyhteydet sen alla toimiville MDT:lle. Kutakin Lustre tiedostojärjestelmää kohti on ainakin yksi MDT joka sisältää tiedostojen metadatan. Kuten MDS, OSS hallinnoi liikennettä sen alla toimivaan OST:iin joita voi olla useita yhtä OSS:ä kohti. Varsinainen tiedostojen sisältämä tieto on tallennettu OST:lle. Lustre-asiakasohjelmisto on rakennettu tarjoamaan yhtenäisen tiedostojärjestelmänäkymän koko Lustre-tiedostojärjestelmään LOV (logical object volume) sekä MDC (metadata client) -rajapintojen kautta. LOV kätkee taakseen kunkin OST:n hallinnoinnin. Näin klusteri ja gridimittakaavassa toimivat laskentaohjelmistot saavat saumattoman pääsyn kaikkiin tarvitsemiinsa tiedostoihin joille niillä on pääsy. Kuten ext4-tiedostojärjestelmissä on Lustre:ssa käyttöoikeudenhallinta, joka perustuu toimivaksi todettuihin POSIX-standardeihin. [WOS+09]

4.3 Kokonaisvaltaisista ratkaisuista

Tieteellisen laskennan resurssitarpeiden kasvaessa on yksityinen klusteri käymässä kalliiksi ja vaikeaksi ylläpitää. Yhä useampi tutkijayhteisö käyttääkin pilvipaleluita jotka ovat nopeasti saatavilla verraten suurella kapasiteetilla kohtalaiseen hintaan. Olemassa olevia klustereita, kuten Ukko-klusteria, ei kuitenkaan sovi unohtaa, vaan perusteellisen resurssien käytön uudelleensuunnittelun myötä niitä voi yhä hyödyntää. Gridit sen sijaan yhä voivat tarjota riittävästi resursseja mutta niiden hallintoalueiden heterogeenisyys voi vähentää niiden muuntautumiskykyä ja siten ehkäistä tehokasta käyttöä. [VPB09]

Pilvipalvelut sen sijaan ovat suunniteltu nimenomaan elastisuutta ajatellen. Virtualisoitu infrastruktuuri voi tarjota käytännössä mitä tahansa resursseja mitä asiakas tarvitsee. Kun resurssien ylläpito keskitetään, on mahdollista rakentaa suurempia resurssikeskittymiä, jotka ovat jaettu käyttäjien kesken myös kustannuksilta. [VPB09]

Määritelmän mukaan pilven infrastruktuuri voi rakentua yhtyeenkytketyistä klustereista gridin tavoin. Klusterin voi siis halutessa muuntaa pilvipalveluksi tarjoamaan laskentaresursseja. Tarkastellaan muutamia ohjelmistokokonaisuuksia jotka mahdollistavat pilvipalveluiden pystyttämisen.

4.3.1 OpenStack

OpenStack on avoimen lähdekoodin projekti jonka tarkoitus rakentaa palvelinkeskukselle käyttöjärjestelmä. Se kattaa laskenta-, verkko- sekä tallennusratkaisuiden keskitetyn hallinnan. Kaikkien resurssien hallinta tapahtuu hallintapaneelin kautta ja automatisoiduilla hallintaskripteillä. Skriptit ja muut hallintaohjelmistot voivat käyttää OpenStack:n laajaa ohjelmointirajapintaa tehtävien hoitamiseen. Hallintaskriptien lisäksi kaikki ohjelmat käyttävät

myös kyseisiä rajapintoja resurssien pyytämiseen OpenStack-pilveltä[18].

OpenStack:ä käytetään pääasiassa palvelinkeskuksen raudan abstrahointiin eli virtuaalisten resurssien jakamiseen IaaS perusteisesti. Erityisesti sitä käytetään toimittamaan julkisia sekä yksityisiä pilvipalveluita [VLDWF12]. OpenStack voidaan konfiguroida monella tavalla, myös toimittamaan jaettuja resursseja, kuten levykuvia nopeaan virtuaalikoneympäristön pystytykseen [ABC+15]. Myös Helsingin Yliopiston Tietojenkäsittelytieteen laitoksella on tutkimuskäytössä OpenStack-laskentaympäristö, joka on pystytetty Ukko-klusterin solmuille 193-240[19]. Monet yritykset tukevat OpenStack-kehitystä [SAE12] ja tarjoavat sille kaupallisia tukipalveluita[20].

4.3.2 Apache Mesos

Mesos ei ole OpenStack:n tyylisesti 'palvelinkeskuksen käyttöjärjestelmä'. Mesos suunniteltiin ja rakennettiin mahdollistamaan usean rinnakkaislaskentaohjelmistokehyksen tehokkaan käytön samassa klusterissa tai gridissä. Sen voi pystyttää esimerkiksi *Amazon EC2* tai OpenStack-yksityispilveen. Mesos on ohut resurssienhallinta kerros, joka tarjoaa siihen kytketyille ohjelmistokehyksille laskentaresursseja, siten sen voidaan nähdä sijoittuvan PaaS-kerrokselle palveluspektriä. ZooKeeper[21] varmistaa Mesos-isännän toiminnan vikatilanteissa ajastamalla varaisännän rikkoutuneen tilalle. Laskentaohjelmiston ohjelmistokehys – kuten Spark – konfiguroidaan käyttämään Mesos-ohjelmointirajapintaa niin, että kun ohjelmistokehykselle tulee tarve ajaa ohjelmaa se pyytää Mesos-isännältä resursseja tarvitsemansa määrän. Isäntä voi vastata pyyntöön sopivalla tarjouksella, jota ohjelmistokehyksen ei ole kuitenkaan pakko hyväksyä vaan se voi odottaa seuraavaa, mahdollisesti parempaa tarjousta. Mesos käyttää siis töiden viivästettyä ajastusta jotta tehtävät saisivat parhaan solmun laskentaan perustuen tiedon lokaliteettiin l. sijaintiin levyllä [ZBSS+10]. [HKZ+11]

Mesos pystyy todistetusti jakamaan resursseja tehokkaasti eri laskentaohjelmistokehysten välillä ja vieläpä tehostamaan laskentaa sekä kapasiteettia [HKZ+11]. Se on myös otettu vastaan monessa kansainvälisesti toimivassa yrityksessä, kuten *eBay*, *Netflix*, *Twitter* sekä *PayPal*[22]. Lisäksi esimerkiksi *Mesosphere* tarjoaa palvelinsalin käyttöjärjestelmää, joka pohjautuu Apache Mesos -järjestelmään[23]. Korkean tason ASF-projektina Mesos-järjestelmää kehitetään jatkuvasti[24].

[18] OpenStack: https://www.openstack.org/ (2015)

[19] Joista ei ole dataa saatavilla, kuten Ukko-klusterin esittelyssä todetaan

[20] OpenStack Marketplace: http://www.openstack.org/marketplace/

[21] Apache ZooKeeper: http://zookeeper.apache.org

[22] Apache Mesos: PoweredBy: http://mesos.apache.org/documentation/latest/powered-by-mesos/ (2015)

[23] Mesosphere: DCOS: https://mesosphere.com

[24] Apache Software Foundation, projektit: http://apache.org/#projects-list

4.3.3 EUCALYPTUS

Eucalyptus on avoimen lähdekoodin projekti jonka tarkoituksena on tarjota *Amazon EC2* -yhteensopiva ohjelmisto pilvipalveluiden pystyttämiseen. Se on erityisesti akateemiseen käyttöön suunnattu, sillä akateemisissa ympäristöissä on monesti klusterityyppinen laskentajärjestelmä saatavilla tutkimukseen. Eucalyptus koostuu neljästä pääkomponentista: Node Controller (NC), Cluster Controller (CC), Cloud Controller(CLC) sekä Storage Controller (Walrus). Nämä yhdessä orkestroivat virtuaalikoneiden ja -verkkojen pystytystä, ylläpitoa ja sulkemista asiakkaiden tarpeiden sekä palveluehtosopimusten puitteissa. Eucalyptus tukee laajaa skaalaa virtuaalikoneita ja virtualisointiohjelmistoja joista merkittävin on KVM[25]. Palveluna Eucalyptus tarjoaa OpenStack-tyylisesti virtualisoitua infrastruktuuria jonka päälle asiakkaat voivat pystyttää haluamansa laskentaympäristön. [NWG+09]

Eucalyptus arkkitehtuurissa CLC on pilven ylin hallinnoija joka toimii sisääntuloväylänä pilveen niin ylläpitäjille kuin käyttäjille. Se päättää korkean tason ajastuksista ja käyttää CC:n tarjoamia palveluita varsinaisten palveluiden käytössä. CC vastaavasti hallinnoi pienempää osa-aluetta: NC-ryhmiä ja virtuaalisia verkkoja. CC:n tehtäviin kuuluu virtuaalikoneiden ajastus NC:lle ja kertoa CLC:lle klusterinsa tilasta. NC on Eucalyptus arkkitehtuurissa pienin osa, jonka tehtävänä on hallinnoida virtuaalikoneita sillä todellisella palvelinresurssilla jolle se on asennettu. [NWG+09]

5 Tieteellinen laskenta Kumpulan kampuksella

Tutkijoita ja professoreita haastateltiin tietojenkäsittelytieteen, kemian ja fysiikan laitoksilta. Heiltä kysyttiin muun muassa: mitä ongelmia he olivat pyrkineet ratkaisemaan, johon he olivat tarvinneet suurta laskentakapasiteettia, mitä ohjelmistokehyksiä ja/tai ohjelmistoja he olivat käyttäneet, miten mainitut ohjelmistokehykset ym. auttoivat ongelmanratkaisussa ja missä ne olivat puutteellisia, minkälaisia ongelmia he kohtasivat laskentaresurssien käytössä ja miten he haluaisivat käyttää niitä.

Yksi kemian laitoksen tutkija kertoi hänen tutkimuksensa keskittyvän ilmakehän nanokemiallisten systeemien ja molekyylien energiatiheyksien mallinnukseen. Hän kertoi käyttäneensä pääasiassa *CSC*:n[26] palveluita. *Gaussian*, *Molpro* sekä *Slurm* -ohjelmistojen ajaminen *Taito*-klusterilla[27] ovat olleet hänen tutkimuksissaan oleellisia.

Tutkijat jotka olivat käyttäneet Ukko-klusteria olivat pyrkineet ratkaisemaan ongelmia lähinnä tietojenkäsittelytieteen alalta, jotkut myös bioinfor-

[25] eucaplyptus yhteensopivuus: https://www.eucalyptus.com/eucalyptus-cloud/iaas/compatibility/4.1

[26] Center for Scientific Computing: https://www.csc.fi

[27] Taito-klusteri on tarkoitettu sarjalaskentaan ja pieniin rinnakkaislaskennan tarpeisiin. Sen etuna on valtava muistin määrä solmua kohden.

matiikan, tilastotieteen ja fysiikan aloilta. Tyypillisiä ohjelmistoja heidän keskuudessaan olivat erilaiset itseohjelmoidut Java-, Python-, Matlab-, ja C++-ohjelmistot. Bioinformatiikkaa tutkivilla suosiossa oli BLAST-ohjelmisto. Pääasiassa ohjelmistot oli käynnistetty useammalle solmuille suorittamaan sarjalaskentaa omilla lähtöarvoilla. Kaikki haastatellut kertoivat, että laskentaympäristön konfigurointi oli ollut haasteellista ja että valmis ympäristö olisi nopeuttanut heidän tutkimustaan. Laskentaresurssien käyttö SecureShell yhteyden ylitse ei muodostanut ongelmia mutta moni olisi mielellään saattanut käyttää graafista käyttöliittymää ohjelmistojen ajon konfigurointiin.

Haastatelluista kaksi oli pystyttänyt Spark-laskentaympäristön käyttöönsä. Ensimmäisen tutkimus liittyi uusien algoritmien suunnitteluun kuvien kohinanpoistoon. Hän oli ajanut kohinanpoistoalgoritmin toteuttavia Spark-ohjelmistoja useille kuville rinnan 80 Spark-solmun laskentaklusterissa. Toisen tutkimus liittyi aihetopologioiden samankaltaisuuksien kartoittamiseen ja reaaliaikaiseen tarjoamiseen verkkohakuja tehdessä. Tähän käytettiin Spark-ympäristöä internetsivujen aihetopologioiden kartoittamisessa.

6 Johtopäätöksiä

Laskentaa käytetään tänä päivänä monella tieteenalalla. Tästä kielii niin kasvava tieteellisten julkaisuiden määrä, jotka käsittelevät laskennalla saavutettuja tuloksia, kuin tieteellisen laskennan vaatimusten kasvu [BAA+06]. Helsingin yliopisto ei ole poikkeus tällä saralla. Suomen opetus- ja kulttuuriministeriön hallinnoima *CSC* tarjoaa laskentakapasiteettia ilmaiseksi korkeakouluille ja tutkijoille.

Tietojenkäsittelytieteenlaitoksen laskentaklusteri Ukko tarjoaa myös laskentaresursseja. Sen käyttö on monimuotoista ja se tarjoaa monelle tutkijalle kapasiteettia heidän tarpeisiinsa. Suppean ohjelmistoinfrastruktuurin vuoksi se kuitenkin kaipaisi uudistusta. Tutkielmassa esitellyt rinnakkaislaskentaohjelmistot voisivat tukea niin tutkijoiden tekemää tutkimusta kuin tietojenkäsittelytieteen laitoksen opetusta. Sellaisten kurssien, kuten BigData frameworks, Biological sequence analysis sekä Overlay and P2P networks järjestäminen helpottuisi kun kurssin järjestäjien ei tarvitsisi jokaisella kurssin iteraatiolla pystyttää laskentaresursseja uudestaan.

Valmiiksi konfiguroidut laskentaohjelmistot helpottaisivat erityisesti tutkijoita, sillä heidän ei tarvitsisi käyttää aikaa ympäristön pystyttämiseen. Myös opiskelijat, jotka haluavat käyttää Ukon laskentakapasiteettia, hyötyisivät ohjelmistoista. Ohjelmistojen resurssivaatimusten ei tarvitse poissulkea tämänhetkistä tapaa käyttää klusterin solmuja – ne voivat toimia rinnan. Olisi myös syytä tarkastella toisiko yksityinen pilviratkaisu Ukolla lisäresursseja. Yksityinen pilvi voidaan rakentaa hyvin skaalautuvaksi ja helppokäyttöiseksi. Tämänhetkinen käyttötapaus muuttuisi vain vähän. Sen sijaan, että käyttäjä kirjautuu todelliselle laskentasolmulle voisi hän käynnistää uuden virtuaaliko-

neen käyttöönsä. Pilven ylläpitoon kuluisi vähemmän aikaa ja käyttökokemus olisi yhtenäisempi ja luotettavampi. Myöskään klusterin kokonaislaskentakapasiteetti ei tällöin alentuisi kun solmuja ei varattaisi yksityiskäyttöön. Klusterin muuttamisessa pilvipalveluksi on huomioitava mahdolliset tulevaisuuden tarpeet. Etenkin akateemisessa ympäristössä on todennäköistä, että järjestelmään tulee asentaa uusia ohjelmistoja. Tämän takia yhteen kaupalliseen ohjelmistoon tai käyttöjärjestelmään lukittautumista tulee pyrkiä välttämään ja siten säilyttää pilvi muuntautumiskykyisenä.

Lähteet

[ABC+15] Abdelrazik, Amr, Bunce, Greg, Cacciatore, Kathy, Mahankal, Sridhar ja Van Rooyen, Frans: *Adding Speed and Agility to Virtualized Infrastructure with OpenStack*. whitepaper, OpenStack, 2015.

[BAA+06] Bernholdt, David E, Allan, Benjamin A, Armstrong, Robert, Bertrand, Felipe, Chiu, Kenneth, Dahlgren, Tamara L, Damevski, Kostadin, Elwasif, Wael R, Epperly, Thomas GW, Govindaraju, Madhusudhan *et al.*: *A component architecture for high-performance scientific computing*. International Journal of High Performance Computing Applications, 20(2):163–202, 2006.

[DG08] Dean, Jeffrey ja Ghemawat, Sanjay: *MapReduce: simplified data processing on large clusters*. Communications of the ACM, 51(1):107–113, 2008.

[Din99] Dincer, Kivanc: *Ubiquitous message passing interface implementation in Java: JMPI*. Teoksessa *Parallel Processing Symposium, International*, sivut 203–203. IEEE Computer Society, 1999.

[FGS06] Fringer, OB, Gerritsen, M ja Street, RL: *An unstructured-grid, finite-volume, nonhydrostatic, parallel coastal ocean simulator*. Ocean Modelling, 14(3):139–173, 2006.

[FZRL08] Foster, Ian, Zhao, Yong, Raicu, Ioan ja Lu, Shiyong: *Cloud computing and grid computing 360-degree compared*. Teoksessa *Grid Computing Environments Workshop, 2008*, sivut 1–10. Ieee, 2008.

[GGL03] Ghemawat, Sanjay, Gobioff, Howard ja Leung, Shun Tak: *The Google file system.* Teoksessa *ACM SIGOPS operating systems review*, nide 37, sivut 29–43. ACM, 2003.

[GHR95] Greenlaw, Raymond, Hoover, H James ja Ruzzo, Walter L: *Limits to parallel computation: P-completeness theory*, nide 200. Oxford university press Oxford, 1995.

[GKS+11] Gopalakrishnan, Ganesh, Kirby, Robert M, Siegel, Stephen, Thakur, Rajeev, Gropp, William, Lusk, Ewing, De Supinski, Bronis R, Schulz, Martin ja Bronevetsky, Greg: *Formal analysis of MPI-based parallel programs.* Communications of the ACM, 54(12):82–91, 2011.

[GL04] Gropp, William ja Lusk, Ewing: *Fault tolerance in message passing interface programs.* International Journal of High Performance Computing Applications, 18(3):363–372, 2004.

[GLDS96] Gropp, William, Lusk, Ewing, Doss, Nathan ja Skjellum, Anthony: *A high-performance, portable implementation of the MPI message passing interface standard.* Parallel computing, 22(6):789–828, 1996.

[HKZ+11] Hindman, Benjamin, Konwinski, Andy, Zaharia, Matei, Ghodsi, Ali, Joseph, Anthony D, Katz, Randy H, Shenker, Scott ja Stoica, Ion: *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.* Teoksessa *NSDI*, nide 11, sivut 22–22, 2011.

[HMH+13] Hussain, Hameed, Malik, Saif Ur Rehman, Hameed, Abdul, Khan, Samee Ullah, Bickler, Gage, Min-Allah, Nasro, Qureshi, Muhammad Bilal, Zhang, Limin, Yongji, Wang, Ghani, Nasir *et al.*: *A survey on resource allocation in high performance distributed computing systems.* Parallel Computing, 39(11):709–736, 2013.

[JSK12] Jakovits, Pelle, Srirama, Satish Narayana ja Kromonov, Ilja: *Stratus: A distributed computing framework for scientific simulations on the cloud.* Teoksessa *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, sivut 1053–1059. IEEE, 2012.

[KCN06] Kohlschütter, Christian, Chirita, Paul Alexandru ja Nejdl, Wolfgang: *Efficient parallel computation of pagerank.* Teoksessa *Advances in information retrieval*, sivut 241–252. Springer, 2006.

[Kos06] Kosar, Tevfik: *A new paradigm in data intensive computing: Stork and the data-aware schedulers.* Genome, 40:50, 2006.

[LM10] Lakshman, Avinash ja Malik, Prashant: *Cassandra: a decentralized structured storage system.* ACM SIGOPS Operating Systems Review, 44(2):35–40, 2010.

[Mat14] Mathew, Sajee: *An overview of Amazon Web Services*, 2014.

[Nas90] Nash, Stephen G. (toimittaja): *A History of Scientific Computing.* ACM, New York, NY, USA, 1990, ISBN 0-201-50814-1.

[NOOY13] Niwa, Junya, Okada, Kazuya, Okuda, Takeshi ja Yamaguchi, Suguru: *MPSDataStore: a sensor data repository system for mobile participatory sensing.* Teoksessa *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, sivut 3–8. ACM, 2013.

[NWG+09] Nurmi, Daniel, Wolski, Richard, Grzegorczyk, Chris, Obertelli, Graziano, Soman, Sunil, Youseff, Lamia ja Zagorodnov, Dmitrii: *The eucalyptus open-source cloud-computing system.* Teoksessa *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, sivut 124–131. IEEE, 2009.

[ORI+10] Oskooi, Ardavan F, Roundy, David, Ibanescu, Mihai, Bermel, Peter, Joannopoulos, John D ja Johnson, Steven G: *MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method.* Computer Physics Communications, 181(3):687–702, 2010.

[PRGK14] Polato, Ivanilton, Ré, Reginaldo, Goldman, Alfredo ja Kon, Fabio: *A comprehensive view of Hadoop research—A systematic literature review.* Journal of Network and Computer Applications, 46:1–25, 2014.

[SAE12] Sefraoui, Omar, Aissaoui, Mohammed ja Eleuldj, Mohsine: *OpenStack: toward an open-source solution for cloud computing.* International Journal of Computer Applications, 55(3):38–42, 2012.

[SK11] Sadashiv, Naidila ja Kumar, SM Dilip: *Cluster, grid and cloud computing: A detailed comparison.* Teoksessa *Computer Science & Education (ICCSE), 2011 6th International Conference on*, sivut 477–482. IEEE, 2011.

[SKP06] Sur, Sayantan, Koop, Matthew J ja Panda, Dhabaleswar K: *High-performance and scalable MPI over InfiniBand with reduced memory usage: an in-depth performance analysis.* Teoksessa *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, sivu 105. ACM, 2006.

[SKRC10] Shvachko, Konstantin, Kuang, Hairong, Radia, Sanjay ja Chansler, Robert: *The hadoop distributed file system.* Teoksessa *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, sivut 1–10. IEEE, 2010.

[SLBA11] Sakr, Sherif, Liu, Anna, Batista, Daniel M ja Alomari, Mohammad: *A survey of large scale data management approaches in cloud environments.* Communications Surveys & Tutorials, IEEE, 13(3):311–336, 2011.

[SYK+10] Seo, Sangwon, Yoon, Edward J, Kim, Jaehong, Jin, Seongwook, Kim, Jin Soo ja Maeng, Seungryoul: *Hama: An efficient matrix computation with the mapreduce framework.* Teoksessa *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, sivut 721–726. IEEE, 2010.

[VLDWF12] Von Laszewski, Gregor, Diaz, Javier, Wang, Fugang ja Fox, Geoffrey C: *Comparison of multiple cloud frameworks.* Teoksessa *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, sivut 734–741. IEEE, 2012.

[VPB09] Vecchiola, Christian, Pandey, Suraj ja Buyya, Rajkumar: *High-performance cloud computing: A view of scientific applications.* Teoksessa *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, sivut 4–16. IEEE, 2009.

[Whi09] White, Tom: *Hadoop: the definitive guide: the definitive guide.* O'Reilly Media, Inc., 2009.

[WMPX11] Wang, Henry, Ma, Yunzhi, Pratx, Guillem ja Xing, Lei: *Toward real-time Monte Carlo simulation using a commercial cloud computing infrastructure.* Physics in medicine and biology, 56(17):N175, 2011.

[WOS+09] Wang, Feiyi, Oral, Sarp, Shipman, Galen, Drokin, Oleg, Wang, Tom ja Huang, Isaac: *Understanding lustre filesystem internals.* Oak Ridge National Laboratory, National Center for Computational Sciences, Tech. Rep, 2009.

[YVCJ07] Yu, Weikuan, Vetter, Jeffrey, Canon, R Shane ja Jiang, Song: *Exploiting lustre file joining for effective collective io*. Teoksessa *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, sivut 267–274. IEEE, 2007.

[ZBSS+10] Zaharia, Matei, Borthakur, Dhruba, Sen Sarma, Joydeep, Elmeleegy, Khaled, Shenker, Scott ja Stoica, Ion: *Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling*. Teoksessa *Proceedings of the 5th European conference on Computer systems*, sivut 265–278. ACM, 2010.

[ZCD+12] Zaharia, Matei, Chowdhury, Mosharaf, Das, Tathagata, Dave, Ankur, Ma, Justin, McCauley, Murphy, Franklin, Michael J, Shenker, Scott ja Stoica, Ion: *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*. Teoksessa *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, sivut 2–2. USENIX Association, 2012.