Clustercalc: Cluster Computing i Regneark

Christian Slot
cslot1@student.aau.dk

11. januar 2019

Aalborg Universitet
Det Tekniske Fakultet for IT og Design
Institut for Datalogi
10th Semester projekt
Vejeder: Bent Thomsen

Summary

In this report I have investigated the use of spreadsheets for cluster computing. Cluster computing is most commonly done via a cluster computing network such as Spark and Hadoop. These networks then often use databases for their data. Both of these are quite hard to use for people without much programming experience.

The accessibility is where spreadsheets are different from most programming languages. They are easier to understand as they are quite close to what you would use if you had to do many calculations by hand. As most of the people who have big amounts of data that they want to process are not fluent in languages such as SQL they have trouble setting up these cluster computing network. This means that they have to explain their business logic to database technicians and computer scientists that then set it up for them. This process i error prone as they often have very different backgrounds and therefore have few common terms with which to communicate.

The idea behind this project is to combine the data processing power of a cluster with the ease of use of spreadsheets. For this goal I have used the open source spreadsheet Funcalc and as the cluster framework I have used AKKA.NET. Both are written in C# which eases the integration and makes it possible to inject the code from the cluster framework directly into the spreadsheet program.

This project report details the the design and implementation of Clustercalc that is a cluster network that uses spreadsheets for both user interface and used for the data processing on the cluster nodes. It enables users to make cluster based calculations while only needing to understand how to use spreadsheets and build in functions. The use of the build in functions is additionally helped by the concept of sheet defined functions that are functions that can be defined in a special function sheet and then used in the rest of the spreadsheet workbook as if they where build in functions. If the user first learns to use the sheet defined functions the leap to using Clustercalcs build in function is not very big.

An additional benefit of using spreadsheets for the calculations on the cluster is that it enables the user to use very big data sets that they normally would not be able to put into spreadsheets. The user can then load the needed subset of information into the user interface spreadsheet while having the calculations and data on the cluster. This is what enables the user to do Big Data calculations as not all the data have to be shown to the user to be used.

Indhold

1 Introduktion

Regneark er et af de mest udbredte programmeringssprog. Det bruges dagligt af mange millioner mennesker verden over til dataanalyse og er så relativt nemt at gå til at det af mange ikke betragtes som et programmeringssprog. Det er særligt udbredt i professionelle sammenhænge inden for bogføring og budgettering og til analyse af samfundsrelateret samt medicinsk data, men bruges også af alt fra folkeskoleelever til privates økonomihåndtering. Regneark er særdeleshed blevet så udbredt, da det er relativt brugervenligt og ikke kræver særligt kendskab til programmering ud over, hvad der er nødvendigt for at kunne bruge en lommeregner. Derudover kan det bruges til at håndtere små såvel som relativt store datamængder. På grund af det høje antal brugere og dets anvendelighed vil der være superbrugere, der bruger det til meget avancerede beregninger med store mængder data. Dermed når nogle superbrugere grænserne for, hvor meget et regneark kan rumme og beregne inden for rimelig tid.

I projektet vil jeg undersøge muligheden for at bruge regnearksprogrammer til Big Data-analyse. Herved kan regnearkssuperbrugerne lave analyser uden at involvere datateknikere og derved undgå dyre databaseløsninger, som Big Data-analyse typisk ender ud i. Man undgår herved også det overhead, der kan opstå, når en superbruger skal forklare terminologier til datateknikerne, inden databasen kan oprettes. Superbrugeren kan i stedet selv sætte beregningerne op i et antal regneark og samle de endelige beregninger i et regneark uden hjælp fra databaseteknikere.

For at muliggøre dette undersøger jeg, hvordan regneark kan distribueres på flere computere og samles til slut i en form for Map Reduce-model. Den primære grund til at distribuere regneark er for at udnytte regnekraft fra flere computere og derved kunne lave flere og større beregninger hurtigere.

Herudover løser distribuering en af de største svagheder ved regneark nemlig håndtering af store datamængder. Med et distribueret system er det muligt at håndtere så store datamængder, som der er plads til samlet på computerne i clusteret, hvilket ikke giver noget teoretisk maksimum på mængden af data. Dette løses ved at have flere regneark med kun en del at dataet i hver, som så har mulighed for at udveksle data med

hinanden, hvis det er nødvendigt. Disse regneark kan således være placeret på forskellige computere i clusteret.

I dette projekt har jeg valgt at anvende en paralleliseret version af Funcalc som mit regnearksprogram. Jeg vil fra nu af omtale dette regnearksprogram som funcalc medmindre det er direkte vigtigt for meningen af differentiere. Det særlige ved Funcalc i forhold til andre regnearksprogrammer er muligheden for at lave »sheet defined functions«, hvilket betyder, at man kan definere funktioner i et særligt regneark kaldet et »function sheet«. Disse funktioner kan herefter kaldes i de almindelige regneark. I forhold til dette projekt har »sheet defined functions« den fordel, at funktioner, som bruges af regnearkene, nemt kan deles mellem alle computerne i clusteret uden at dele selve dataregnearkene. Ligeledes virker de også som en skabelon for, hvordan man bruger den tilføjede funktionalitet og letter overgangen, hvis man allerede kan bruge »sheet defined functions« og forstår indbyggede funktioner.

Der er ikke mange der har beskæftiget sig med regneark som programmeringsmodel og endnu færre der har beskæftiget sig med parallelisering af disse på et cluster. Det tætteste på et cluster baseret regneark er i ExcelGrid af Nadiminti et al. [4]. ExcelGrid bruger et cluster af computere med en virtuel computer oven på, hvilket gør at de kan bruge regnekraften på clusteret til at lave beregningerne fra regnearket. På den måde ligger det tæt på det dette projekt udforsker men det løser ikke problemet med plads i regneark hvor denne løsning ikke kan håndtere mere en Excel som det er et plugin til. Wack [10], Abramson et al. [1] beskriver hvorfor regnearksmodellen er en god kandidat til parallelisering og eksempler på hvordan det kan gøres og deres principper kan godt bruges i cluster sammenhæng.

1.1 Problemformulering

Jeg vil undersøge hvordan regnearksprogram kan blive paralleliseret i et cluser af computere ved hjælp af Actor modellen. Jeg vil herefter undersøge om og i hvilken grad det er muligt at bruge regnearksprogrammer til Big Data analyse.

- Kan regneark parralleliserers på et cluster?
- Kan regneark bruges til Big Data analyse?

2 Relateret arbejde

2.1 Spreadsheet Implementation Technology

Sestoft [8] beskriver i denne bog implementationen af Funcalc og hans overvejelser omkring denne. Desuden indeholder den også eksempler på hvordan det internt virker og hvilke modeller og forskning det er baseret på.

2.2 Stream Processing with a Spreadsheet

Vaziri et al. [9] har lavet et værktøj til at visualisere »streaming data« i regneark for derved at gøre det nemmere for data analytikere uden så stor programmeringsforståelse at fremstille regneark, som løbende henter data og opdaterer celler. Analytikerne kan gennem regnearket så lave grafer og udregninger på samme måde som de normalt gør i Excel med den forskel at regnearket hele tiden bliver opdateret med ny data.

2.3 ActiveSheets: Super-Computing with Spreadsheets

Abramson et al. [1] har i denne artikel beskrevet, hvordan det er muligt at parallelisere beregninger i regneark uden brugerne mister funktionalitet og brugervenlighed. De gør dette ved at lave et værktøj, der henter dataet fra regnearket, paralleliserer udregningerne backend og så sendes tilbage til Excel.

2.4 ExcelGrid: A .NET Plug-in for Outsourcing Excel Spreadsheet Workload to Enterprise and Global Grids

Nadiminti et al. [4] har lavet et plug in til Excel, der gør det muligt at sende beregningerne i regnearket til et »grid«, der så laver udregningerne. Gennem dette »grid« kan beregningerne blive sendt til en virtuel computer, der trækker på computerne i en form

for cluster, og resultatet bliver sendt tilbage til Excel. Herved får man en meget større beregningskraft men bevarer illusionen om, at det kører på en computer.

2.5 Online partial evaluation of sheet-defined functions

I denne artikel beskriver Sestoft [7], hvordan »sheet defined functions« fungerer, og hvordan de er implementeret i Funcalc. »Sheet defined functions« gør det muligt for regnearksbrugeren selv at lave noget, der ligner de indbyggede funktioner der er i regnearksprogrammer. Dette er gjort muligt ved at have en speciel form for ark, hvor man kan definere funktioner næsten, som hvis det var en almindelig regnearksberegning. Herefter er det så muligt at bruge disse funktioner på de almindelige ark.

2.6 Partitioning Dependency Graphs for Concurrent Execution: A Parallel Spreadsheet on a Realistically Modeled Message Passing Environment

Wack [10] beskriver som en af de første, hvordan beregningerne fra et regneark kan paralleliseres, og hvorfor regneark er en god model for parallelisering. I selve implementationen har han brugt det funktionelle programmeringssprog Scheme, og det giver dem mulighed for at have brugerdefinerede funktioner via Schemes lambda expressions eller gennem noget, de kalder »rules«. Rules er en matematisk beskrivelse af funktionen, hvor de eventuelle underfunktioner bliver placeret i deres egen celle. Ifølge Wack er grunden til at regneark er særligt godt til parallelisering, at hver celle ikke har nogen sideeffekter, som man ellers kæmper meget med i forhold til parallelisering af andre programmeringsmodeller.

2.7 CUDAfy og Corecalc

Pedersen [5] beskriver i denne rapport, hvordan man kan bruge GPUen til at lave beregninger i Corecalc. Han gør dette ved at tilføje en indbygget funktion, der fungerer som interface imellem Corecalc og hans program, der så bruger GPUen til at lave beregningerne.

2.8 Analyzing spreadsheets for parallel execution via model checking

Bøgholm et al. [3] er en kort rapport, der beskriver resultaterne af “Popular Parallel Programming”-projektet. Her vises det at det er muligt at finde ud af hvordan regnearksberegninger skal paralleliseres på den bedste måde men at det ved nogen svære beregninger bliver dyrt.

3
Baggrund

3.1 Big Data

Big data er ikke et specielt veldefineret begreb. Der er mange meninger om hvad begrebet Big data skal dække. Det anvendes oftest til at beskrive data, der er for stort eller på anden måde ikke passer ind i de løsninger, der allerede eksisterer. Man kan forestille sig en situation, hvor der ikke er plads til alt dataet i en database, hvorfor det er nødvendigt at have flere databaser, der kan håndtere en samlet datamængde. I dette projekt har jeg derfor søgt at belyse, hvordan regneark kan anvendes til at bearbejde store datamængder, som ellers ikke kan rummes i regneark.

3.2 Corecalc, Funcalc og Puncalc

Corecalc er et regnearksprogram skrevet af Sestoft og som er beskrevet i"A Spreadsheet Core Implementation in C#[6]". Denne er skrevet på baggrund af at der ikke eksisterede nogle open source regnearksprogrammer, der var velegnede til forskning. Hermed menes, at de enten var closed source, såsom Excel, eller meget dårligt dokumenteret såsom Libre Office. Som en overbygning på Coreaclc har Sestoft lavet Funcalc[8], der tilføjer "Sheet defined functions" i en speciel form for regneark, der så kan bruges i resten af projektmappen. Næste skridt var at udvikle Puncalc, der er en parallel udgave af Funcalc. Puncalc dækker over mange udviklingsgrene på GitHub med forskellige funktionaliteter, men som alle kan køre deres beregninger parallelt. I dette projekt har jeg brugt Mastergrenen af projektet, da den er den mest testede og stabile.

3.3 Parallelisering af regneark

Mange programmer lavet i andre programmeringssprog har problemer med at parallellisere databehandlingen. Dette skyldes, at deres programmeringsmodeller har svært ved at undgå sideeffekter og dermed gør dataflowet kompliceret.

Regneark har derimod en mere simpel programmeringsmodel, da hver celle kun kan referere til et antal celler eksplicit og man derfor altid kan lave en support-graf for beregningerne, hvilket giver et mere simpelt dataflow. Denne support-graf kan så bruges til at se, hvilke beregninger der er uafhængige af hinanden og dermed kan paralleliseres. Yderligere kan regnearksprogrammet via disse grafer tjekke for cykliske referencer imellem cellerne.

3.4 AKKA.NET

AKKA.NET er en C# implementation af Actormodellen. Actormodellen er en model for »concurent« databehandling via en Actor. Actors er universelle primitiver, hvilket betyder, at i Actormodellen er alting Actors. Disse Actors kan sende beskeder mellem hinanden, som så behandles i den rækkefølge de modtages i. Eftersom hver besked behandles separat i den rækkefølge de er modtaget i, er der ingen sideeffekter og ikke behov for data locks.

3.5 AKKA.NET eller MS Orleans

MS Orleans er Microsoft Researchs bud på en implementation af Actormodellen. De har valgt at lave en mere restriktiv model for Actors, de kalder Virtual Actor Model[2]. Denne model har den fordel at have implementeret mange af de ting, man almindeligvis gerne vil sikre, såsom »atleast once messaging«. Dette har den ulempe, at man ikke selv har indflydelse på, hvordan det er implementeret, og at man også bruger regnekraft på de ting, man ikke har brug for. I dette projekt har jeg valgt at bruge AKKA.NET da jeg ikke vidste om jeg ville bruge nogen af de ting som Virtual Actor modellen ikke understøtter. Herudover forventede jeg ikke jeg ville få brug for de bekvemmeligheder som MS Orleans tilføjer.

4 Design

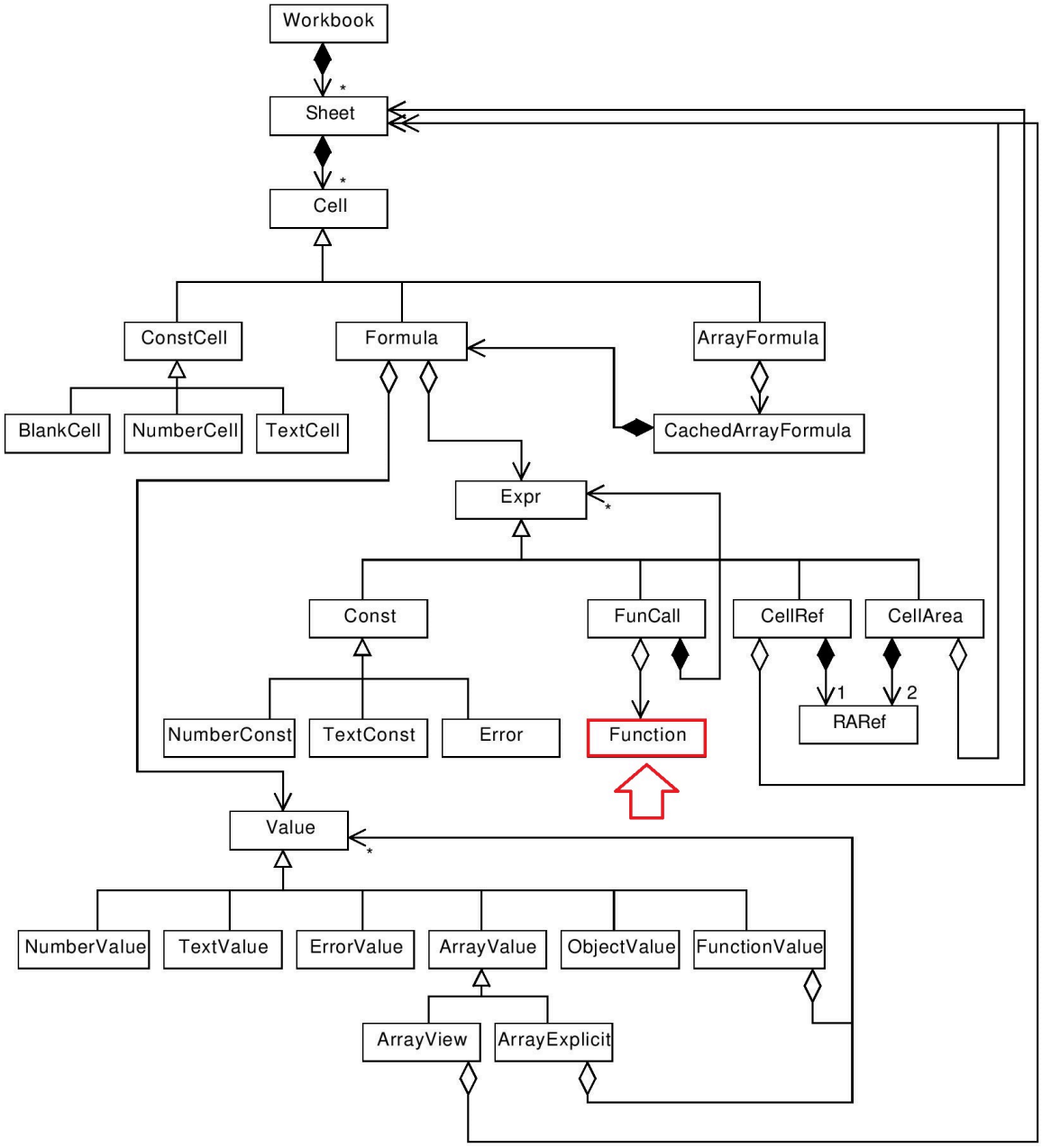
I dette kapitel beskriver jeg designet af Clustercalc. Iden bag Clustercalc er at bruge regneark til at behandle store datamængder via et cluster af regneark.

4.1 AKKA cluster

Ved brug af AKKA er det naturligt at have en lagdelt struktur. I dette projekt har jeg valgt at have tre lag. Øverst er et manager-lag, der står for kommunikationen mellem clusteret og regnearket. Under dette er der brug for et lag, der håndterer hver enkelt computer på clusteret. Nederst er så selve arbejderne på clusteret. Disse står for at lave alle beregningerne. som skal sendes op til brugergrænsefladen. Derudover håndterer de også dataet på clusteret. Som beregningsmedie bruger hver af disse arbejdere en Funcalc Workbook, som laver selve beregningerne.

4.2 Indbygget funktion

Kommunikationen mellem clusteret og det primære regnearksprogram, der agerer GUI for løsningen, skal være forståelig for en regnearkssuperbruger. Jeg har derfor valgt at bruge en lignende fremgangsmåde som Pedersen [5]. Fremgangsmåden er at tilføje en indbygget funktion til Funcalc, der fungerer som interface mellem GUIen og de enkelte regneark på clusteret. Funktionaliteten bliver altså tilføjet i Funktion i figur 4.1 på næste side.



Figur 4.1: Funcalc klasse hierarki.

5 Teknisk rapport

I dette kapitel er det beskrevet hvordan implementationen fungerer.

5.1 Actor systemet generelt

Til at oprette de forskellige nodes på clusteret har jeg brugt Topshelf der sikrer at de bliver oprettet på samme måde. Et eksempel på dette ses i kodereference 5.1. I dette eksempel kan man se TopShelf lave en Windows service på baggrund af MachineManagerens Console Application. Denne har en Start og en Stop funktion der hver især håndterer hvad der sker når serviceen starter, altså den opretter actorsystemet og MachineManager actoren, og når den stoppes, actorsystemet tremineres. TopShelf er et bibliotek der gør det nemmere at lave Windows services. Konceptet bag Topshelf er at det håndterer interaktionen mellem programmørens "service logic" og den indbyggede service support der er i .Net.

```
40          [STAThread]
41          static void Main(string[] args)
42          {
43              Console.SetWindowSize(180, 20);
44
45              HostFactory.Run(configure =>
46              {
47                  configure.Service<MachineManagerService>(
                        service =>
48                  {
49                      service.ConstructUsing(s => new
                            MachineManagerService());
50                      service.WhenStarted(s => s.Start());
51                      service.WhenStopped(s => s.Stop());
```

```
52                      //continue and restart directives are also
                            available
53                  });
54
55                  configure.RunAsNetworkService();
56                  configure.UseAssemblyInfoForServiceInfo();
57
58              });
59          }
60      }
```

Listing 5.1: Brug af Topshelf i main

5.2 Actor model for systemet

Actormodellen, jeg har brugt i dette projekt har tre primære lag.

Nederst er WorkbookActor-laget. Det er WorkbookActor laget, der laver alle udregningerne og databehandlingen, der skal laves på clusteret. Dette gør de ved at oprette og manipulere deres indbyggede Workbook.

Over disse ligger Machine laget, der er en abstraktion for de enkelte computere i clusteret. Machine Actorene styrer det arbejde, der skal laves på hver maskinem ved at tilføje WorkbookActors.

Øverst er MachineManager-laget, der agerer som interface mellem Funcalc-regnearket og clusteret. MachineManageren er dermed den første knude i clusteret som Machine laget sender beskeder til.

Herudover er det en ekstra node der agerer "Seed node" for clusteret. Denne bliver håndteret af s værktøjet fra Petabridge. Lighthouses primære funktion er at hjælpe med interaktionen mellem AKKA.NET clusteret og PaaS services så som Asure. I projektet har jeg brugt det som en meget letvægtig node der kun agerer seed node.

5.2.1 MachineManager

MachineManageren har to primære opgaver. Den første er at agere interface mellem Funcalc GUIen og clusteret. Den anden er at holde styr på de enkelte Machine Actors i clusteret og deres Actor-referencer. Der kunne potentielt være mere end en MachineManager i clusteret, altså flere regneark med GUI der bruger de samme maskiner og WorkbookActor's med mindre tweeks i HOCON-filen, der er en konfigurationsfil AKKA.NET bruger til at oprette Actoren. Som det er implementeret i Clustercalc sender actorene i Machine laget en besked til en prædefineret lokation, når de bliver initialiseret, hvilket gør, at der kun kan være en MachineManager.

5.2.2 Machine

Machine Actoren håndterer beskeder mellem MachineManagerens Funcalc GUI og WorkbookActorne og holder styr på de enkelte WorkbookActors og deres referencer, så beskeder let kan sendes videre til den rette. Da den bliver initialiseret på de enkelte computere i clusteret, er det også den, der sørger for at de enkelte WorkbookActors bliver oprettet lokalt på de enkelte computere.

5.2.3 WorkbookActor

WorkbookActoren er den, der håndterer selve beregningerne i clusteret. Man kan lave så mange WorkbookActors, som man vil, på hver Machine, og hver WorkbookActor håndterer en Funcalc Workbook. Det er så i disse Funcalc Workbooks, at beregningerne bliver lavet.

WorkbookActoren har en række beskeder, som den kan modtage, der får den til at manipulere dens Workbook. Når selve WorkbookActoren bliver oprettet, gør den det med en Workbook, som også giver navn til WorkbookActoren. På den måde er det garanteret at hver Workbook har et unikt navn, som bruges når data skal hentes fra denne.

5.3 Interface mellem Funcalc og Clustercalc

MachineManager agerer som interface mellem Funcalc og Clustercalc-projektet. Det gør den ved at tilføje en funktion til Functions klassens funktionsliste, hvilket gør at Funcalc herefter ser den som en indbygget funktion. Denne funktion er så den, der bruges til at kommunikere med clusteret. Koden, der bliver tilføjet til funktionslisten og nu kan kaldes som en indbygget funktion, er i kodelisting 5.2. Funktionen bruges ved at kalde funktionen i MachineManagerens Funcalc instans via “=ClusterCalc” og så de to input for funktionen i parrentes med et keyword først, der siger, hvad man gerne vil gøre og så et input til dette eg. “(“get”, “machine1:workbook1:sheet1:A1”)”. Samlet vil det så skrives “=Clustercalc(“get”, “machine1:workbook1:sheet1:A1”)” hvilket betyder, at den spørger om værdien i cellen A1 på sheet1 i workbook1 på machine1.

```
67      private static ActorSystem mySystem;
68      private static IActorRef manager;
69
70      private long RequestId = 0;
71      // =CLUSTERCALC("GET", "Machine:Workbook:Sheet:Cell")
72      public Value ClusterAccess(TextValue command, TextValue
            reference)
73      {
74        if (command is TextValue && reference is TextValue)
75        {
76          string Command = command.ToString(), Reference =
                reference.ToString();
```

```
77
78          if (Command.ToUpper() == "GET")
79          {
80            string[] SplitReference = Reference.Split(':');
81            RequestId = RequestId + 1;
82            Task<RespondCellValue> value;
83
84            if (SplitReference.Length == 4)
85            {
86              var Cell = new Funcalc.Corecalc.Addressing.CellAddr
                  (SplitReference[3]);
87              value = manager.Ask<RespondCellValue>(new
                  GetCellValueFromMachineWorkbook(RequestId,
                  SplitReference[0], SplitReference[1],
                  SplitReference[2], Cell.col, Cell.row));
88            }
89            else
90            {
91              return ErrorValue.argCountError;
92            }
93
94            if (double.TryParse(value.Result.Value, out double
                NumberResult))
95            {
96              return NumberValue.Make(NumberResult);
97            }
98            else
99            {
100             return TextValue.Make(value.Result.Value);
101           }
102         }
103         else if (Command.ToUpper() == "INIT")
104         {
105           string folderPath;
106
107           if (Reference == "")
108           {
109             folderPath = "D:/Studie/funcalc-examples-master/
                  applied/";
110           }
111           else
112           {
113             folderPath = Reference;
```

```
114             }
115
116             var ReturnedMachineActorRefs = manager.Ask<
                    ReturnMachineActorRef>(new GetMachineActorRef(
                    RequestId++));
117             IActorRef[] MachineActorRefs =
                    ReturnedMachineActorRefs.Result.MachineActorRef;
118
119             var WorkbookLoaderActor = mySystem.ActorOf(
                    WorkbookLoader.Props(), "WorkbookLoader");
120             WorkbookLoaderActor.Tell(new RequestLoadWorkbooks(
                    RequestId, folderPath, MachineActorRefs));
121
122             return TextValue.FromString("MachineManager␣
                    initialized␣on␣cluster");
123           }
124           else
125           {
126             return TextValue.FromString("did␣not␣understand");
127           }
128         }
129         else
130         {
131           return ErrorValue.argTypeError;
132         }
133       }
```

Listing 5.2: Den tilføjede funktion

6 Diskussion

6.1 Programmeringssprog

I dette projekt har jeg valgt at skrive i programmeringssproget C#. Det er valgt på baggrund af at jeg ville bruge Funcalc (Puncalc) som basis for mit program og det er skrevet i C#. Da jeg gerne ville bruge Actormodellen som baggrund for distribueringen af beregninger og data, har jeg brugt AKKA.NET. AKKA.NET er en C# implementation af Actormodellen baseret på en Java/Scala implementation.

Det meste af AKKA.NET er programmeret af programmører et firma, der hedder Petabridge, og bærer præg af, at de gerne ville have så meget af funktionaliteten oversat til C#, men ikke har været så grundige med dokumentationen. Specielt ved de mere specialiserede funktioner er mange af forklaringskommentarerne TBD, hvilket gør det svært at finde ud af, hvordan funktionerne skal anvendes. Det var dog en stor fordel at kunne holde hele projektet i C#, da det gjorde det nemmere at tilføje funktionalitet direkte i Funcalc. Herudover er debugging også nemmere, specielt i forhold til det tilføjede kode i Funcalc. Det er dog stadig en ulempe ved C#, at der ikke er så stærke open source-fællesskaber, og man dermed ikke altid kan finde et vedligeholdt bibliotek til det, man har brug for. En anden stor fordel ved at holde alt koden inden for .NET er, at jeg kunne bruge den samme IDE gennem hele projektet, og at Visual Studio er meget stærkt udviklingsværktøj.

6.2 Problemer under programmeringen

Jeg har brugt lang tid på problemer med deling af information om de forskellige nodes i clusteret. Der er hjælpebiblioteker i AKKA.NET projektet, såsom DistributedPubSubMediator-projektet, der kan bruges til at håndtere dette. DistributedPubSubMediator-projektets funktionalitet er, at man kan sende beskeder mellem actors, der er abonnerer på et emne, uden at kende deres sti eller actor reference. Disse var dog meget dårligt beskrevet, og jeg kunne ikke få det til at virke. Dette kan skyldes den måde, jeg havde valgt at opbygge

clusteret, at hjælpebiblioteket ikke var opdateret til at virke med den nyest version af AKKA.NET, eller at jeg ikke forstod, hvordan det skulle bruges. Dette resulterede i, at jeg brugte en mere low level løsning hvor Machine actorene sender en besked direkte til MachineManager actoren via en predefineret sti til actoren.

6.3 Generelle resultater

Under programmeringen af den indbyggede funktion, der tilføjes til Funcalc, overvejede jeg to tilgange. Den ene er at tilføje en funktion til hver måde at tilgå clusteret, fx en Get-funktion, der giver værdien af en celle på clusteret og en Set-funktion, der ændrer en celleværdi på clusteret. Den anden tilgang er kun at tilføje en enkelt funktion, som så bruger et keyword til at bestemme, hvad den skal gøre med resten af inputtet, fx en Clustercalc funktion, der så har to input, et keyword såsom GET og så en sti til den celle hvis værdi skal hentes.

Jeg har til projektet forsøgt at lægge mig op af, hvordan Excel trækker data ind fra andre Excel-ark for at holde det i samme stil og derfor have en vis genkendelighed for Excel-superbrugere. Til mit møde med Lars Krull blev det dog hurtigt klart, at selvom han brugte regneark meget til forskellig dataanalyse med rimelig store datamængder, brugte han ikke Excels mere avancerede funktioner. Så selvom min måde at tilgå Workbooks på clusteret lægger sig op af Microsofts måde at tilgå andre regneark på computeren, vil det nok være ukendt for mange slutbrugere.

6.4 Fordele og ulemper ved implementationen

Fordelen ved at have al data-udregning i regneark og så hente resultater fra disse til et samlings-regneark er, at det giver brugeren en forståelse af hvad der sker i clusteret. Brugeren vil selv kunne rette i regnearkene, der bliver brugt på clusteret og dermed have forståelse for, hvad der bliver udregnet. Dette er i kontrast til Nadiminti et al. [4], hvor de trækker dataet ud af et Excel ark og så sender det til beregning på en virtuel computer, der trækker på en form for cluster til beregningskraften. Eftersom beregningerne er blevet afkoblet fra cellerne i regnearket vil de i teorien bedre kunne blive load balanced og derfor kunne køres mere effektivt. Nadiminti et al. [4] er dog kun et svar på, hvordan man kan løse problemet med beregningskraften for et enkelt regneark og ikke en løsning på, hvordan man løser problemet med plads i regneark.

7
Konklusion

I denne rapport har jeg undersøgt hvordan Actormodellen kan bruges til parallelisering af regnearksprogrammer. Jeg har fundet at regneark er en simpel men meget ekspressiv model for MapReduce lignende cluster computing hvor man samler resultatet af beregningerne på et bruger regneark. Desuden gør løsningen det muligt for regnearksbrugere at parallelisere beregningen af store datamængder uden forståelse der ligger meget uden for almindelig brug af regneark.

Jeg har fundet at det er muligt at parallelisere beregninger og databehandling fra et regneark på et cluster kun ved hjælp af "message pasing" og regneark til beregning på cluster noderne.

I dette projekt har jeg fundet at det er muligt at lave håndtere "Big Data" datamængder ved hjælp af regneark på et cluster da de enkelte regneark ikke behøver at indeholde alt dataet.

8 Fremtidigt arbejde

I dette kapitel beskriver jeg fremtidigt arbejde for projektet.

8.1 Load balancing af workbooks

I dette projekt består load balancing af at dele Workbooks ligeligt mellem de antal maskiner, der er tilgængelige på clusteret. Dette er en meget simpel måde at load balance som ikke giver meget garanti om at det er balanceret medmindre de forskellige Workbooks er balancerede. Hvis der kunne findes måde at sammenligne de forskellige Workbooks ville det være muligt at balancere dem bedre.

8.2 Brugertest

En af de største fordele ved regneark er hvor nemme de er at bruge for slutbrugerne. Det er dog ikke sikkert at det er intuitivt den måde jeg har valgt at implementere brugergrænsefladen på. En bruger test vil derfor give vigtige input til hvordan brugen af interface regnearket kan forbedres.

8.3 Streaming af data

Da det i denne løsning er brugerne der laver alle regneark også dem på clusteret er det en fordel at de alle kun behøver at være plaseret på en computer. Dette betyder dog at disse regneark skal senders over clusteret til de enkelte WorkbookActors. Problemet ved dette er at med større regneark bliver det dyrt at sende det over netværket og fordi så lang tid misser actorene Heartbeat beskeder og bliver måske afslutte eller sat til "unreachable". TCP og UDP er også optimerede til at sende mange mindre pakker i stedet for få store hvilket går ud over performance. Løsningen på dette er at streame dataet ved først at lave objektet om til et array af bytes, dele det op i mindre pakker og så sende dem til destinations actoren som så laver det om til objektet igen. Eftersom det ikke tager

så lang tid at sende hver enkelt pakke kan actoren også svare på heartbeat beskeder indimellem og derfor undgå at blive lukket ned.

Litteratur

[1] David A Abramson, Paul Roe, Lew Kotler, and Dinelli Mather. Activesheets: super-computing with spreadsheets. In Adrian Tentner, editor, *Grand Challenges in Computer Simulation: Proceedings of the High Performance Computing Symposium - HPC 2001*, pages 110 – 115, United States, 2001. The Society for Modeling & Simulation International. ISBN 1-56555-237-7. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.9599&rep=rep1&type=pdf`.

[2] Philip A. Bernstein, Sergey Bykov, Alan Geller, Gabriel Kliot, and Jorgen Thelin. Orleans : Distributed virtual actors for programmability and scalability. 2014.

[3] Thomas Bøgholm, Kim G. Larsen, Marco Muniz, Bent Thomsen, and Lone Leth Thomsen. Analyzing spreadsheets for parallel execution via model checking. In *Essays on the Occasion of Bernhard Steffen's 60th Birthday (Lecture Notes in Computer Science, vol. 11200)*. Springer-Verlag, 2018.

[4] Krishna Nadiminti, Yi-Feng Chiu, Nick Teoh, Akshay Luther, Srikumar Venugopal, and Rajkumar Buyya. Excelgrid: A.net plug-in for outsourcing excel spreadsheet workload to enterprise and global grids. In *Proceedings of the 12th International Conference on Advanced Computing and Communication*, 12 2018. URL `http://pirun.ku.ac.th/~fengjtp/JOS_jos201023.pdf`.

[5] Niels Brøndum Pedersen. Cudafy og corecalc. Master's thesis, Aalborg University, 2016. URL `https://projekter.aau.dk/projekter/files/239502303/report.pdf`.

[6] Peter Sestoft. A spreadsheet core implementation in c#. *The IT University of Copenhagen*, 2006.

[7] Peter Sestoft. Online partial evaluation of sheet-defined functions. In *Festschrift for Dave Schmidt*, 2013. URL `https://pdfs.semanticscholar.org/6d0f/087cc77f47006858d18cce916e23d6ad0840.pdf`.

[8] Peter Sestoft. *Spreadsheet Implementation Technology: Basics and Extensions*. The MIT Press, 2014. ISBN 0262526646, 9780262526647.

[9] Mandana Vaziri, Olivier Tardieu, Rodric Rabbah, Philippe Suter, and Martin Hirzel. Stream processing with a spreadsheet. In *Proceedings of the 28th European Conference on ECOOP 2014 — Object-Oriented Programming - Volume 8586*, pages 360–384, New York, NY, USA, 2014. Springer-Verlag New York, Inc. ISBN 978-3-662-44201-2. doi: 10.1007/978-3-662-44202-9_15. URL `http://dx.doi.org/10.1007/978-3-662-44202-9_15`.

[10] Andrew P. Wack. *Partitioning Dependency Graphs for Concurrent Execution: A Parallel Spreadsheet on a Realistically Modeled Message Passing Environment.* PhD thesis, University of Delaware Newark, DE, USA, Newark, DE, USA, 1996. UMI Order No. GAX96-10479.