Weiterentwicklung eines Task Distribution Frameworks

Bachelor-Thesis von Georg Heesch
Februar 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
p2p

Weiterentwicklung eines Task Distribution Frameworks

Vorgelegte Bachelor-Thesis von Georg Heesch

1. Gutachten: Benjamin Schiller
2. Gutachten: Prof. Dr. Thorsten Strufe

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 17. Februar 2014

(Georg Heesch)

Inhaltsverzeichnis

Zusammenfassung

Viele zusammen können das schaffen, was der Einzelne nicht vermag.

Dieses Zitat von IG BAU-Bundesvorstandsmitglied Carsten Burckhardt trifft nicht nur auf Arbeiterverbände zu, sondern auch auf die Maschinen, die uns heutzutage viel Rechenarbeit abnehmen. Viele, wenn auch nicht alle, Aufgaben für Rechenmaschinen lassen sich in Teilaufgaben unterteilen, Dadurch werden Probleme die für einzelne Maschinen schlicht zu groß, aufwändig oder langwierig sind, lösbar.

Doch ebenso wie sich Arbeiter in Gewerkschaften und Interessengemeinschaften zusammenschliessen um ein gemeinsames Ziel zu verfolgen, müssen auch Maschinen von Systemen zur Aufgabenverwaltung koordiniert werden.

Eines dieser Systeme wird in dieser Arbeit erweitert und verbessert.

1 Einleitung

Viele Aufgaben sind auch heute noch zu groß um mit Alltagsrechenmaschinen sinnvoll bearbeitet zu werden. Meist sind die Aufgaben so komplex, dass der schiere Rechenaufwand handelsübliche Systeme überfordert oder es bedarf zur Bearbeitung einem Ausmaß an Ressourcen mit dem auch spezialisierte Systeme nicht dienen können.

Ein Ansatz dieses Problem zu Lösen sind so genannte “Supercomputer”, gigantische Rechenmaschinen, die an Leistung und/oder vorhandenen Ressourcen normale Rechner um ein Vielfaches übertreffen. Ein anderer Ansatz ist es die Aufgaben in kleine Teilaufgaben aufzuteilen und diese von getrennten Rechnern bearbeiten zu lassen.

Keiner der beiden Ansätze ist für jedes Problem geeignet, da auch Supercomputer ihre Grenzen haben und nicht jede Aufgabe sich parallelisieren lässt.

Ein Task Distribution Framework versucht letzteren Ansatz zu vereinfachen, indem es die Verteilung der Aufgaben und das Sammeln der Ergebnisse übernimmt.

Ein solches System wurde 2012 von Jan Dillmann im Rahmen seiner Bachelorarbeit[2] geschaffen

Andere verbreitete Task Distribution Frameworks sind Apaches Hadoop[4] und Boinc[1]. Beide sind jedoch auf eine andere Klasse von Problemen ausgelegt.

Während das System von Jan Dillmann für sehr bandbreitenintensive Aufgaben gedacht war, ist Boinc eher auf sehr berechnungslastige Probleme ausgerichtet und Hadoop ist auf die Bearbeitung grosser Datenmengen spezialisiert.

Wie jedes System werden natürlich auch Task Distribution Systeme mit einem oder mehreren bestimmten Problemen im Hinterkopf entworfen, daher ist es jedem selbst überlassen sich darüber klar zu werden, welches System für sein Problem das beste ist.

2 Problemstellung

2.1 Bestehendes System

Leider sind viele dieser Frameworks auf Entwickler ausgelegt und bieten nicht die Möglichkeit die Verteilung der Aufgaben komplett aus dem Design der Software herauszuhalten, oder gar bereits existierende Software zu verwenden.

Aus der Abschlussarbeit von Jan Dillmann[2] heraus entstand ein System zum verteilten Abarbeiten getrennter, unabhängiger Aufgaben.

Ich wurde gebeten, dieses System zu erweitern und benutzbarer zu machen.

Das System besteht aus drei grundlegenden Komponenten: einem Server, einer Clientsoftware und dem Worker, der die Berechnung ausführt.

Der Server ist dergestalt aufgebaut, dass eine Verwaltungssoftware Tasks in einer Datenbank ablegt, auf die die Clients zugreifen. Jeder Task enthält Informationen über die Bezugsquelle für Software, die den Worker darstellt sowie die Parameter mit denen diese Software aufgerufen werden soll. Außerdem enthält die Datenbank auch Informationen darüber wie viel Zeit dem Worker gelassen werden soll. Die Software muss in einer von Jan Dillmann definierten Art bereitgestellt sein, sodass zusammen mit der Software auch Informationen über das Umfeld in dem Sie ausgeführt werden soll, bezogen wird.

Der Client bezieht aus der Datenbank die nötigen Informationen und aus der dort beschriebenen Bezugsquelle den Worker. Anschließend richtet er das Umfeld für den Worker ein und führt diesen mit den vorgegebenen Parametern aus. Die Parameter werden dabei nicht auf der Kommandozeile übergeben, sondern in einer Textdatei übergeben. Dies ermöglicht die Übergabe einer größeren Menge an Informationen an den Worker in einem übersichtlicheren Format als reine Kommandozeilenparameter es ermöglichen.

2.2 Anforderungen

Da das bestehende System noch nicht über ein Userinterface verfügte, sollte eine Sammlung an Kommandozeilenwerkzeugen geschaffen werden um die Interaktion mit dem System zu vereinfachen.

Es wurde die Möglichkeit des Loggings gewünscht, und auf Basis dieser Logs sollten statistische Untersuchungen durchgeführt werden und in einfach lesbarer Art präsentiert werden.

2.3 Usecases

Einem Nutzer soll es mit wenig Aufwand möglich sein, seine Tasks in das Task Distribution Framework einzuarbeiten.

Er soll zu diesem Zwecke seine Tasks in einer Textdatei definieren und über Kommandozeilenwerkzeuge in das System bringen. Nach Abarbeitung seiner Tasks sollen ihm Statistiken über den Verlauf zur Verfügung stehen und angemessen präsentiert werden.

3 Design und Implementierung

3.1 Designentscheidungen

Ich wollte versuchen, die Änderungen am bestehenden System so gering wie möglich zu halten.

Für das Logging habe ich mich entschieden, eine möglichst Simple Lösung zu verwenden.

Ein getrennter Daemon sollte einen Port bereitstellen und die eingehenden Strings mit einem Timestamp versehen und in eine Datei schreiben. Die Logs sollten regelmäßig rotiert werden.

An diesen Port können die Clients ihre Nachrichten richten. Die Nachrichten sind in einem Simplen CSV Format gehalten.

Die Logs werden von einer Statistiksoftware ausgewertet und die Ergebnisse von einer Rendersoftware aufbereitet.

Die Statistik- und Rendersoftware sollte Modular aufgebaut sein und leicht zu erweitern sein.

Das Userinterface besteht wunschgemäß aus mehreren Kommandozeilenwerkzeugen, die zur besseren Automatisierung ausschließlich auf Textdateien arbeiten.

3.2 Software

3.2.1 Server Actions

AddNamespace

Erstellt einen Namespace in der Datenbank und fügt ihn zur Liste der Namespaces hinzu.

Der Namespace wird durch Übergabe einer Textdatei im Json Format definiert. diese kann mit Dateinamen als Parameter übergeben werden oder direkt nach stdin gegeben werden. der Namespace ist dabei ein JSON-Object das zumindest das Feld Name hat.

Mehrere Namespaces können in einem JSON-Array zusammengefasst werden.

```
1  echo '{"name": "example"}' | AddNamespace
```

Listing 3.1: Beispiel eines Aufrufs von AddNamespace

AddTask

Erstellt einen Task in der Datenbank und fügt ihn zum set „new“ seines respektiven namespaces hinzu.

Der Task wird durch Übergabe einer Textdatei im Json Format definiert. diese kann mit Dateinamen als Parameter übergeben werden oder direkt nach stdin gegeben werden. der Task ist dabei ein JSON-Object das zumindest die Felder NameSpace und Worker hat.

Mehrere Tasks können in einem JSON-Array zusammengefasst werden.

Wenn eine ID übergeben wird, werden die angegebenen Felder des existierenden Tasks überschrieben.

```
1 echo '{"NameSpace": "example","Worker": "file:///data/worker.zip"}' | AddTask
```

Listing 3.2: Beispiel eines Aufrufs von AddTask

DeleteNamespace

Entfernt einen Namespace und alle seine Sets, Listen, Tasks und Tasklisten

Der Namespace wird durch Übergabe einer Textdatei im Json Format definiert. diese kann mit Dateinamen als Parameter übergeben werden oder direkt nach stdin gegeben werden. der Namespace ist dabei ein JSON-Object das zumindest das Feld Name hat.

Mehrere Namespaces können in einem JSON-Array zusammengefasst werden

```
1 echo '{"name": "example"}' | DeleteNamespace
```

Listing 3.3: Beispiel eines Aufrufs von DeleteNamespace

DeleteTask

Entfernt einen Task aus allen Listen und Sets eines Namespaces und entfernt den Task aus der Datenbank

Der Task wird durch Übergabe einer Textdatei im Json Format definiert. diese kann mit Dateinamen als Parameter übergeben werden oder direkt nach stdin gegeben werden. der Task ist dabei ein JSON-Object das zumindest die Felder NameSpace und ID hat.

Mehrere Tasks können in einem JSON-Array zusammengefasst werden.

```
1 echo '{"NameSpace": "example","ID": "123"}' | DeleteTask
```

Listing 3.4: Beispiel eines Aufrufs von DeleteTask

DeleteTaskList

Entfernt eine TaskListe aus allen Listen und Sets eines Namespaces und entfernt die TaskListe aus der Datenbank

Die TaskListe wird durch Übergabe einer Textdatei im Json Format definiert. diese kann mit Dateinamen als Parameter übergeben werden oder direkt nach stdin gegeben werden. Die TaskListe ist dabei ein JSON-Object das zumindest die Felder NameSpace und ID hat.

Mehrere TaskListen können in einem JSON-Array zusammengefasst werden.

```
1 echo '{"NameSpace": "example","ID": "123"}' | DeleteTasklist
```

Listing 3.5: Beispiel eines Aufrufs von DeleteTasklist

ExportProcessed

Gibt alle abgeschlossenen Tasks in einem JSON Format aus

Requeue

Sammelt alle abgelaufenen Tasks, sowie alle Tasks des Sets „new“ in Listen und reiht diese in die Warteschlange ein.

Show

Gibt ein angegebenes Element aus.

Die Eingabe erfolgt über eine Textdatei im JSON-Format und die Objekte werden dabei in einem JSON-Array aufgelistet, selbst wenn es nur ein Objekt gibt.

die Objekte werden durch einen String mit dem Aufbau „tdf.<namespace>“ für Namespaces und
„tdf.<namespace>.<typ>.<id>“ für Tasks und Tasklisten identifiziert

```
1 echo '["tdf.example.task.123","tdf.example.tasklist.23"]' | Show
```

Listing 3.6: Beispiel eines Aufrufs von Show

3.2.2 JSON Format

Die Darstellung eines Tasks in für Mensch und Maschine lesbarer Form haben wir duch Verwendung des JSON Formats erreicht.

Jeder Task ist ein JSON Objekt mit unten aufgelisteten Feldern.

Zeiten sind in Millisekunden

```
1  {
2          "ID": "1234",
3          "NameSpace": "examples",
4          "Session": "1970-1-1:123456"
5          "Worker": "http://example.com/download/worker.zip",
6          "Input": "123 fad 456 bar",
7          "RunBefore": "Mon  9 Dec 12:00:00 CET 2013",
8          "RunAfter": "Mon  9 Dec 00:00:00 CET 2013",
9          "Timeout": 60000,
10         "WaitAfterSetupError":1000,
11         "WaitAfterRunError":1000,
12         "WaitAfterSuccess":1000
13 }
```

Listing 3.7: Beispiel eines Tasks im Json Format

3.2.3 Logserver

Ursprünglich habe ich in Erwägung gezogen auf das weit verbreitete Syslog Protokoll[3] zurückzugreifen, mich jedoch dafür entschieden eine erheblich simplere Lösung zu wählen, da wir viele der Möglichkeiten eines Syslog nicht brauchen und damit eine unnötige Abhängigkeit geschaffen wird.

Für den Logserver habe Ich ein kleines Javaprogramm geschrieben, dass den Port öffnet und bei eingehender Verbindung einen String entgegen nimmt und in eine Datei schreibt. Der Pfad

dieser Datei wird dabei für jeden String in Abhängigkeit von der System-zeit und einem Konfigurationsparameter neu generiert, sodass beispielsweise jede Stunde automatisch eine neue Datei erzeugt wird, wenn eine Nachricht eingeht. Wenn keine Nachricht eingeht wird auch keine Datei erzeugt.

3.2.4 Statistik

Die Statistikkomponente ist wie geplant modular aufgebaut. dabei gibt es zwei Arten von Modulen.

- Mathmodules
- Rendermodules

Wrapper

Die Module werden von einem in ruby geschriebenen Wrapper koordiniert. Dieser wird über eine Textdatei im JSON Format konfiguriert, die die Module, ihre Eingabedateien, Ausgabedateien und das Intervall, in dem die Module ausgeführt werden, angibt. Ein Beispiel ist in Listing 3.8.

Die Reihenfolge der Definitionen spielt dabei keine Rolle, jedoch werden die Module in der Reihenfolge ausgeführt, in der sie im „Modules“ Array aufgelistet sind. Dies ist bedingt durch die Definition von Arrays und Hashes in JSON.

Der Wrapper wertet dabei den Asterisk als Wildcard aus und ersetzt dabei einige Strings in den Dateinamen durch von der System-zeit abhängige Strings. Siehe dazu Tabelle 3.1.

%s	Unix Timestamp, sekundengenau
%d	Tag des Monats
%m	Monat als Nummer
%B	Monat als Text
%Y	Jahr, vierstellig
%y	Jahr, zweistellig
%F	Datum, äquivalent zu „%Y-%m-%d“
%z	Zeitzone „±hhmm“
%Z	Zeitzone als Text, abgekürzt
%M	Minute
%H	Stunde (24h)
%%	%

Tabelle 3.1: Formatierungsoptionen Dateinamen

```
{
  "logfiles": [
    "/path/to/Log*"
  ],
  "modules": [
    "module1",
    "module2",
    "module3"
  ],
  "definitions": {
    "module1": {
      "executable": "mathmodule/module1.sh",
      "output": "module1.%s.xml",
      "input": [
        "logs"
      ],
      "interval": 600
        },
        "module2": {
      "executable": "mathmodule/module1.sh",
      "output": "module2.%s.csv",
      "input": [
        "logs"
      ],
      "interval": 600
        },
        "module3": {
      "executable": "renderer/module3.sh",
      "output": "module3.%s.png",
      "input": [
        "module1.%s.xml",
        "module2.%s.csv"
      ],
      "interval": 600
        }
  }
}
```

Listing 3.8: example Wrapper Configuration

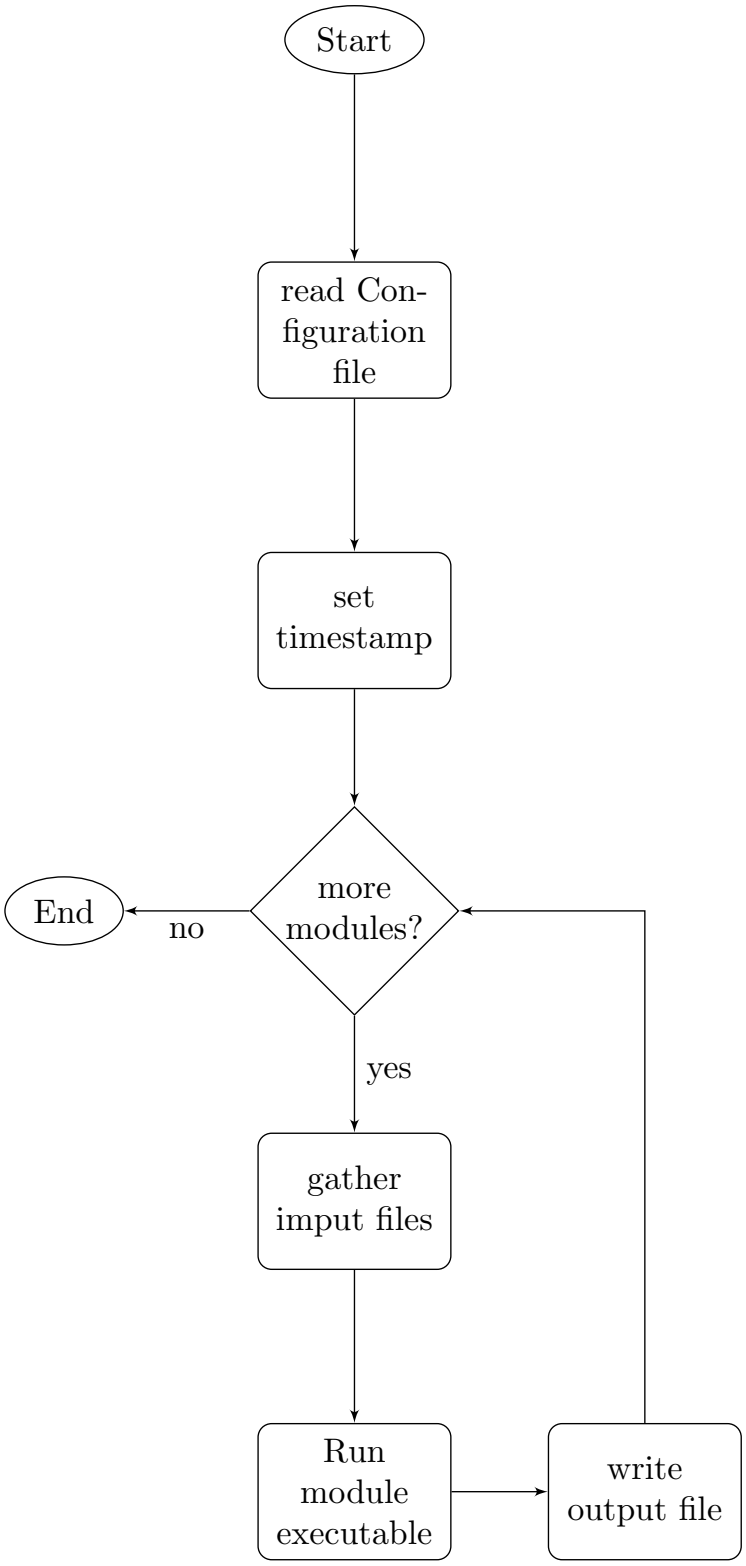


Abbildung 3.1: Workflow des Wrappers

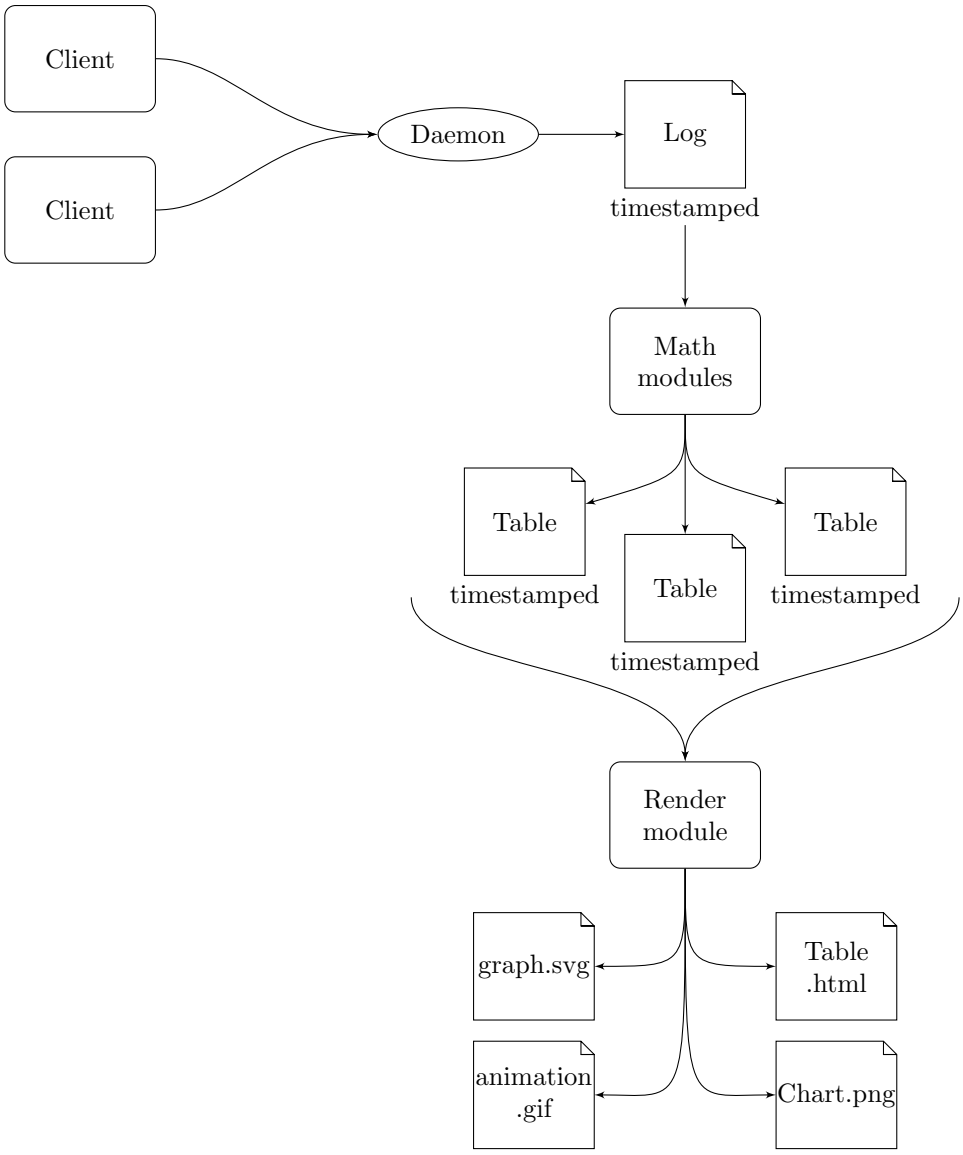


Abbildung 3.2: Datenfluss im Logging

Modulaufbau

Damit die Module miteinander kommunizieren können, müssen sie sich natürlich an einige Vorschriften halten.

Jedes Modul ist eine Executable, die auf dem System ohne weitere Rücksichtnahme direkt ausgeführt werden kann. Im Speziellen sollen keine Interpreter aufgerufen werden müssen. Dies lässt sich unter unixartigen Systemen leicht erreichen indem dem zu interpretierenden Script eine Interpreterzeile[1] vorrangestellt wird.

Dem Modul werden als Parameter ausschließlich die Dateinamen auf denen es operieren soll übergeben.

Die Ergebnisse des Moduls werden auf der Standardausgabe erwartet und vom Wrapper in eine entsprechende Ausgabedatei geschrieben.

Dadurch sind die Module dahingehend eingeschränkt, dass jedes Modul genau eine Datei erzeugen kann.

Mathmodules

Ein Mathmodule zeichnet sich dadurch aus, dass es nicht unbedingt Menschen-lesbaren Output hat.

Ein Beispielhaftes Modul sehen Sie in Listing 3.9

[1] #! /interpreterpfad

```
 1 #!/bin/bash
 2
 3 #Folgendes Script ermittelt die Anzahl der Fehlgeschlagenen Tasks
 4 # in den übergebenen Logfiles
 5
 6 unset hash
 7 declare -A hash
 8
 9 cat "$@" | while IFS="," read timestamp type host content; do
10         if [ "type" = "Task_Fail" ] ; then
11                 hash[$host] = $((${hash[$host]:-0} + 1))
12         fi
13 done
14 for i in ${!hash[@]}; do
15         echo $i ${hash[$i]}
16 done
```

Listing 3.9: Beispielhaftes Mathmodule

Rendermodules

Ein Rendermodule operiert nicht direkt auf den Logfiles sondern auf dem Output eines oder mehrerer Mathmodules. Die Aufgabe eines Rendermodules beschränkt sich darauf die Daten für den Benutzer lesbar zu präsentieren.

3.3 Logging

Das Logging beruht auf einigen wenigen Nachrichten, die von jedem Client für jeden Task und jede Liste abgesetzt werden. eine Kurzreferenz ist in Tabelle 3.3 zu finden

Der Client beginnt damit dem Logging seine Existenz zu verkünden (Client_init) Bei regelkonformem Beenden des Clients wird auch dies mitgeteilt.

Das Leben eines Tasks beginnt aus Sicht des Loggings mit dem Task_Claim, der Übernahme der Verantwortung für einen Task durch den Client. Ein Task_Init signalisiert den Beginn der Vorbereitungen für einen Task. Sobald der Task dann tatsächlich bearbeitet wird, sendet der Client ein Task_Start. Zu diesem Zeitpunkt sind alle Vorarbeiten erledigt. Sobalt der Task erfolgreich abgeschlossen wurde sendet der Client ein Task_Success

Schlägt ein Task fehl, oder kann er nicht erfolgreich vorbereitet werden, sendet der Client ein Task_Fail. Den Unterschied zwischen den beiden Möglichkeiten erkennt man im Log daran ob der task vor oder nach der Nachricht Task_Start fehlschlägt

3.3.1 Interfaces

Logging
ServerActions
Datenbankdesign:

```
 1  1391782224510,Client_Init,Client1
 2  1391782224543,Tasklist_Claim,Client1,example.123, example.72, example.73,
       example.74
 3  1391782360122,Task_Init,Client1,example.72
 4  1391782398203,Task_Fail,Client1,example.72
 5  1391782321996,Task_Init,Client1,example.73
 6  1391782322066,Task_Start,Client1,example.73
 7  1391782359078,Task_Fail,Client1,example.73
 8  1391782399244,Task_Init,Client1,example.74
 9  1391782399304,Task_Start,Client1,example.74
10  1391782435318,Task_Success,Client1,example.74
```

Listing 3.10: Beispielhafte Abläufe von Tasks im Log

Client Init	Signalisiert die Initialisierung eines Clients, erste Logmessage jedes Clients
Claim Tasklist	Liefert einen eindeutigen Identifier der Liste Sowie der Tasks in der Liste, passiert nach erfolgreichem Auschecken einer Tasklist
Task Fail	Liefert den eindeutigen Identifier des Tasks, passiert nach fatalen Fehlern im Ablauf des Tasks (ob setup oder Ausführung)
Task Init	liefert den eindeutigen Identifier des Tasks, signalisiert erfolgreichen Abschluss des Setups
Task Success	liefert den eindeutigen Identifier des Tasks, signalisiert erfolgreichen Abschluss des Tasks, der Task ist damit abgeschlossen
Client shutdown	Letzte Logmessage jedes Clients bei ordnungsgemäßem shutdown

Tabelle 3.2: die Nachrichtentypen des Loggings

tdf.namespaces	Array	Eine Liste der Vorhandenen Namespaces
tdf.{namespace}.task.{id}	Hash	Eigenschaften eines Tasks
tdf.{namespace}.tasklist.{id}.Tasks	Array	die Tasks in einer Taskliste
tdf.{namespace}.new	Array	neue Tasks, die noch nicht eingereiht wurden.

Tabelle 3.3: keys in der Datenbank

4 Implementation

4.1 Änderungen an vorhandenen Klassen und Funktionen

Der Client wurde dahingehend modifiziert, dass er statt einzelner Tasks nun Listen von Tasks abarbeitet und somit seltener auf die Datenbank zugreifen muss.

4.2 Neuer code

Es wurde eine neue Klasse zur Repräsentation der Tasklisten hinzugefügt, deren methoden sich an der Klasse für Tasks orientieren.

Die Server Actions wurden implementiert und ein minimalistischer Server für das Loging geschaffen.

Es wurde ein Wrapper geschrieben, der die Module zum Berechnen der Statistiken koordiniert sowie einige Module zum Auswerden der Logs.

5 Evaluation

5.1 Testsetup

Zu Testzwecken wurde eine Szenario entwickelt, dass ein über mehrere Computer verteilten Testrahmen für Studenten bietet.

Dazu werden Programme, die die Studenten einreichen in Tasks verpackt und in das Task Distribution Framework gegeben.

5.2 scripte zum erzeugen der Testdaten

Das folgende Listing zeigt die setup.sh.head, die später zum Generieren der Workerpakete verwendet wird.

```
1  #!/bin/bash
2
3  datasection=‘grep -n --text ’^exit #data starts in line below$’ $0|head -n1 |
       cut -d: -f1‘
4
5  script=‘realpath $0‘
6
7  cd $3
8  tail -n +$((datasection+1)) $script | tar x
9
10 exit #data starts in line below
```

Das damit entstehende Shellscript entpackt die in sich selbst enthaltenen Daten der Studierenden in den vom Framework vorgegebenen Temporären Ordner, wo sie dem Worker zur Verfügung stehen.

Von den Studenten wird erwartet, dass sie ein ausführbares Script mit dem Name „student-run.sh“ zur Verfügung stellen, da dies den Anfangspunkt für den Worker darstellt. Der Worker selbst führt dieses Script dann aus.

```
1  #!/bin/bash
2
3  infile=$1
4  outfile=$2
5  tmpdir=$3
6
7  executable=./student-run.sh
8
9  cd $tmpdir && $executable < $infile >$outfile
```

Folgendes Script generiert aus Ordnern, die die Executables der Studenten enthalten ein Worker Paket nach den Vorschriften von Jan Dillmann und eine JSON-Datei die zur verwendung mit AddTask geeignet ist.

```
1  #!/bin/bash
2  function gentask() {
3  echo -n ’{
```

```
 4 "NameSpace": "'${2:-Students}'",
 5 "Session": "Students-2021-4-25",
 6 "Worker": "file:///root/worker-'$1'.zip",
 7 "Input": "",
 8 "Timeout": 60000,
 9 "WaitAfterSetupError":100,
10 "WaitAfterRunError":100,
11 "WaitAfterSuccess":10
12 }'
13 }
14 function gentasks(){
15 while [ "$1" ]; do
16 (
17 cp setup.sh.head  setup.sh
18 chmod +x $1/*.sh
19 tar -c -C $1 . >> setup.sh
20 7z a -Tzip worker-${1%/}.zip setup.sh run.sh
21
22 ) > /dev/null
23 gentask ${1%/} ${1%-*}
24 shift
25 [ "$1" ] && echo , || echo
26 done
27 }
28
29 echo \[
30 gentasks */
31 echo \]
```

5.3 Module und Ergebnisse

5.3.1 wrapper.conf

```
 1 {
 2   "logfiles": [
 3     "../../testsetup/student-cluster/Log*"
 4   ],
 5   "modules": [
 6     "Task_Duration",
 7     "claims_successes_fails_per_host",
 8                 "claims_successes_fails_per_namespace",
 9     "claims_successes_fails_global",
10     "fails_per_host.svg",
11     "fails_per_host.pie",
12                 "fails_per_minute",
13                 "fails_per_minute-graph",
14     "fails_per_host.html"
15   ],
16   "definitions": {
17     "Task_Duration": {
18       "executable": "mathmodule/Task_duration.sh",
19       "output": "Task_Duration.%s",
```

```
20        "input": [
21          "logs"
22        ],
23        "interval": 600
24      },
25      "claims_successes_fails_global": {
26        "executable": "mathmodule/claims_successes_fails_global.sh",
27        "output": "claims_successes_fails_global.%s",
28        "input": [
29          "claims_successes_fails_per_host.%s"
30        ],
31        "interval": 1200
32      },
33      "fails_per_host.svg": {
34        "executable": "renderer/fails_per_host.sh",
35        "output": "fails_per_host.%s.svg",
36        "input": [
37                                  "claims_successes_fails_per_host.%s"
38        ],
39        "interval": 1200
40      },
41      "fails_per_host.pie": {
42        "executable": "renderer/fails_per_host.pie.sh",
43        "output": "fails_per_host.pie.%s.svg",
44        "input": [
45                                  "claims_successes_fails_per_host.%s"
46        ],
47        "interval": 1200
48      },
49      "fails_per_host.html": {
50        "executable": "renderer/fails_per_host.html.sh",
51        "output": "fails_per_host.%s.html",
52        "input": [
53                                  "fails_per_minute.%s.svg",
54                                  "claims_successes_fails_per_host.%s",
55                                  "fails_per_host.%s.svg",
56                                  "fails_per_host.pie.%s.svg"
57        ],
58        "interval": 1200
59      },
60      "claims_successes_fails_per_host": {
61        "executable": "mathmodule/claims_successes_fails_per_host.sh",
62        "output": "claims_successes_fails_per_host.%s",
63        "input": [
64          "logs"
65        ],
66        "interval": 600
67      },
68      "claims_successes_fails_per_namespace": {
69        "executable": "mathmodule/claims_successes_fails_per_namespace.sh",
70        "output": "claims_successes_fails_per_namespace.%s",
71        "input": [
72          "logs"
73        ],
74        "interval": 600
```

```
75      },
76      "fails_per_minute": {
77        "executable": "mathmodule/fails_per_minute.sh",
78        "output": "fails_per_minute.%s",
79        "input": [
80          "logs"
81        ],
82        "interval": 600
83      },
84      "fails_per_minute-graph": {
85        "executable": "renderer/linegraph.sh",
86        "output": "fails_per_minute.%s.svg",
87        "input": [
88          "fails_per_minute.%s"
89        ],
90        "interval": 600
91      }
92    }
93  }
```

5.3.2 Worker

sleepy.sh

```
1  #!/bin/bash
2
3  #sleepy  dwarf: sleeps (possibly too) long and succeeds
4
5  sleep $((RANDOM%60+45))
6
7  true
```

sneezy.sh

```
1  #/!bin/bash
2
3  #sneezy  dwarf: sleeps and sometimes fails
4
5  sleep $((RANDOM%30+30))
6
7  [ $((RANDOM%3)) -gt 0 ]
```

happy.sh

```
1  #!/bin/bash
2
3  #happy dwarf: sleeps and succeeds
4
5  sleep $((RANDOM%5 +15))
6
7  true
```

bashful.sh

```
#/!bin/bash

#bashful  dwarf: sleeps and fails

sleep $((RANDOM%30+30))

false
```

dopey.sh

```
#/!bin/bash

#dopey  dwarf: sleeps and fails dependent on caffeine level...

sleep $((RANDOM%30+30))

m=‘date +%M‘

fail=$((20 - (m%20)))

[ $((RANDOM%fail)) -gt 0 ]
```

5.3.3 fails_over_time.sh.inc

```
function fails_over_time(){

logfile=$1
interval=$2
host=$3

tmpfile=‘mktemp‘

cat $logfile > $tmpfile

head -n1 $tmpfile | cut -d \) -f2 | cut -d , -f1 | read firstevent

start=$((firstevent/interval))

a=""
typeset -A a

grep -i task_fail $tmpfile | while read event; do

echo $event | cut -d \) -f2 | cut -d , -f1 | read time
index=$(((time/interval)-start))
a[$index]=$((a[$index]+1))

done

stop=$index
```

```
27 index=0
28
29 while [ $index -le $stop  ]; do
30 echo $index $((a[$((index++))]))
31 done
32 rm $tmpfile
33
34 }
```

5.3.4 fails_per_host.sh

```
 1 #!/bin/zsh
 2 source ‘dirname $0‘/uniq.sh
 3 source ‘dirname $0‘/fails_over_time.sh.inc
 4
 5 tmpfile=‘mktemp‘
 6 outfile=‘basename ${0} .sh‘.‘date +%s‘
 7 #while [ "$2" ]; do
 8 #shift
 9 #done
10 cat "$@" > $tmpfile
11
12 fails_over_time $tmpfile 60000
13
14 rm $tmpfile 2>/dev/null
```

5.3.5 linegraph.sh

```
 1 #!/bin/zsh
 2
 3
 4 function plotparts(){
 5 n=${1:-0}
 6 while [ $n -gt 0 ]; do
 7         echo -n "\"$tmpfile\" using $((n+1)) title \"client $((n--))\" , "
 8 done
 9
10 }
11
12 tmpfile=‘mktemp‘
13
14 cat "$@" > $tmpfile
15
16 n=‘head -n2 $tmpfile | tail -n1 | wc -w | cut -d "      " -f2‘
17 #set output \"test.svg\"
18
19 (echo -n "set terminal svg size 1200,600 fname 'Verdana' fsize 10
20 set style fill   solid 1.00 border lt -1
21 set xtics border in scale 1,0.5 nomirror rotate by -45 offset character 0, 0,
       0
22 set datafile missing '-'
23 set style data histogram
24 set style histogram
```

```
25 set style fill solid border -1
26 set boxwidth 0.75
27 set key outside;
28 plot [-1:] [0:] \"$tmpfile\" using 4:xticlabels(1) notitle"
29 echo
30 ) | gnuplot
31 rm $tmpfile
32 exit
```

5.3.6 fails_per_minute.sh

```
 1 #!/bin/zsh
 2
 3 outfile="test.png"
 4 tmpfile=‘mktemp‘
 5
 6 function nop(){}
 7
 8 cat "$@" > $tmpfile
 9
10 (echo 'set term svg size 600,400
11
12 unset offsets
13 set style fill solid 1.0 border -1
14 set offsets 0.5, 0.5, 1, 0
15 min_y = GPVAL_DATA_Y_MIN
16 max_y = GPVAL_DATA_Y_MAX
17 mean(x) = mean_y
18 fit mean(x) "'$tmpfile'" using 1:2 via mean_y
19 plot [0:] [0:]  mean_y with lines lc rgb "#0f0f0f" title "mean", "'$tmpfile'"
       using 1:2 with lines lc rgb "#f01050" title ""
20 '
21 )| gnuplot 2>/dev/null
22
23
24 rm $tmpfile
25 exit
```

5.3.7 claims_successes_fails_global.sh

```
 1 #!/bin/zsh
 2 source ‘dirname $0‘/uniq.sh
 3
 4 tmpfile=‘mktemp‘
 5 outfile=‘basename ${0} .sh‘.‘date +%s‘
 6 cat "$@" > $tmpfile
 7
 8 delim=" "
 9
10 total_claim=0
11 total_success=0
12 total_fail=0
13
```

```
14 cut -d"$delim" -f2,3,4 $tmpfile | while read claim success fail ; do
15 total_claim=$((total_claim+claim))
16 total_success=$((total_success+success))
17 total_fail=$((total_fail+fail))
18
19
20 done
21 echo $total_claim $total_success $total_fail
22
23
24 rm $tmpfile
```

5.3.8 claims_successes_fails_per_host.sh

```
 1 #!/bin/zsh
 2 source ‘dirname $0‘/uniq.sh
 3
 4 tmpfile=‘mktemp‘
 5 outfile=‘basename ${0} .sh‘.‘date +%s‘
 6 cat "$@" > $tmpfile
 7
 8 delim=","
 9
10 cut -d"$delim" -f3 $tmpfile | uniq | while read host ; do
11 echo -n "$host "
12 grep $host$delim $tmpfile | grep -ic Task_Claim | tr -d "\n"
13 echo -n " "
14 grep $host$delim $tmpfile | grep -ic Task_Success | tr -d "\n"
15 echo -n " "
16 grep $host$delim $tmpfile | grep -ic Task_Fail
17
18 done | sort
19
20
21 rm $tmpfile
```

5.3.9 claims_successes_fails_per_namespace.sh

```
 1 #!/bin/zsh
 2 source ‘dirname $0‘/uniq.sh
 3
 4 tmpfile=‘mktemp‘
 5 outfile=‘basename ${0} .sh‘.‘date +%s‘
 6 cat "$@" > $tmpfile
 7
 8 delim=","
 9
10 cut -d"$delim" -f4 $tmpfile | cut -d . -f1 | uniq | sort -h -k 1.7 | while
       read host ; do
11          if [ "$host" ]; then
12                  echo -n "$host "
13                  grep $host. $tmpfile | grep -ic Task_Claim | tr -d "\n"
14                  echo -n " "
```

```
15                  grep $host. $tmpfile | grep -ic Task_Success | tr -d "\n"
16                  echo -n " "
17                  grep $host. $tmpfile | grep -ic Task_Fail
18          fi
19  done
20
21
22  rm $tmpfile
```

5.3.10 Task_duration.sh

```
 1  #!/bin/zsh
 2  source ‘dirname $0‘/uniq.sh
 3
 4  tmpfile=‘mktemp‘
 5  outfile=‘basename ${0} .sh‘.‘date +%s‘
 6  cat "$@" > $tmpfile
 7
 8
 9  delim=’,’
10  cut -d"${delim}" -f4 $tmpfile | uniq | while read task ; do
11  #grep $task $tmpfile | grep -i Task_Claimed | cut -d"${delim}" -f1 | read
        claimed
12  taskinfo=‘grep ${delim}$task’\(’${delim}’\|$\)’ $tmpfile | sed ’s:$:\\\\n:’ ‘
13  #taskinfo=‘grep ${delim}$task’$’ $tmpfile | sed ’s:$:\\\\n:’ ‘
14  completed=‘echo -e $taskinfo| grep -i Task_Success | cut -d"${delim}" -f1 |
        cut -d’)’ -f2‘
15  claimed=‘echo $taskinfo | grep -i Task_Init  | cut -d"${delim}" -f1 | cut -d’)
        ’ -f2‘
16
17
18  [ "$completed" ] && [ "$claimed" ] && echo $task $((completed - claimed))
19  done #| sort
20
21
22  rm $tmpfile
```

5.3.11 task_to_host.sh

```
 1  #!/bin/zsh
 2  source ‘dirname $0‘/uniq.sh
 3
 4  tmpfile=‘mktemp‘
 5  outfile=‘basename ${0} .sh‘.‘date +%s‘
 6  cat "$@" > $tmpfile
 7
 8
 9
10  grep -i task $tmpfile | cut -d, -f4,3 --output-delimiter=" " | uniq
11
12  rm $tmpfile
```

5.3.12 uniq.sh

```
function uniq(){
read array
echo $array
while read line; do

if [ "$line" ] && ! echo $array|grep -q "^$line\$"; then
array="$array
$line"
echo $line
fi
done

}
```

5.4 output

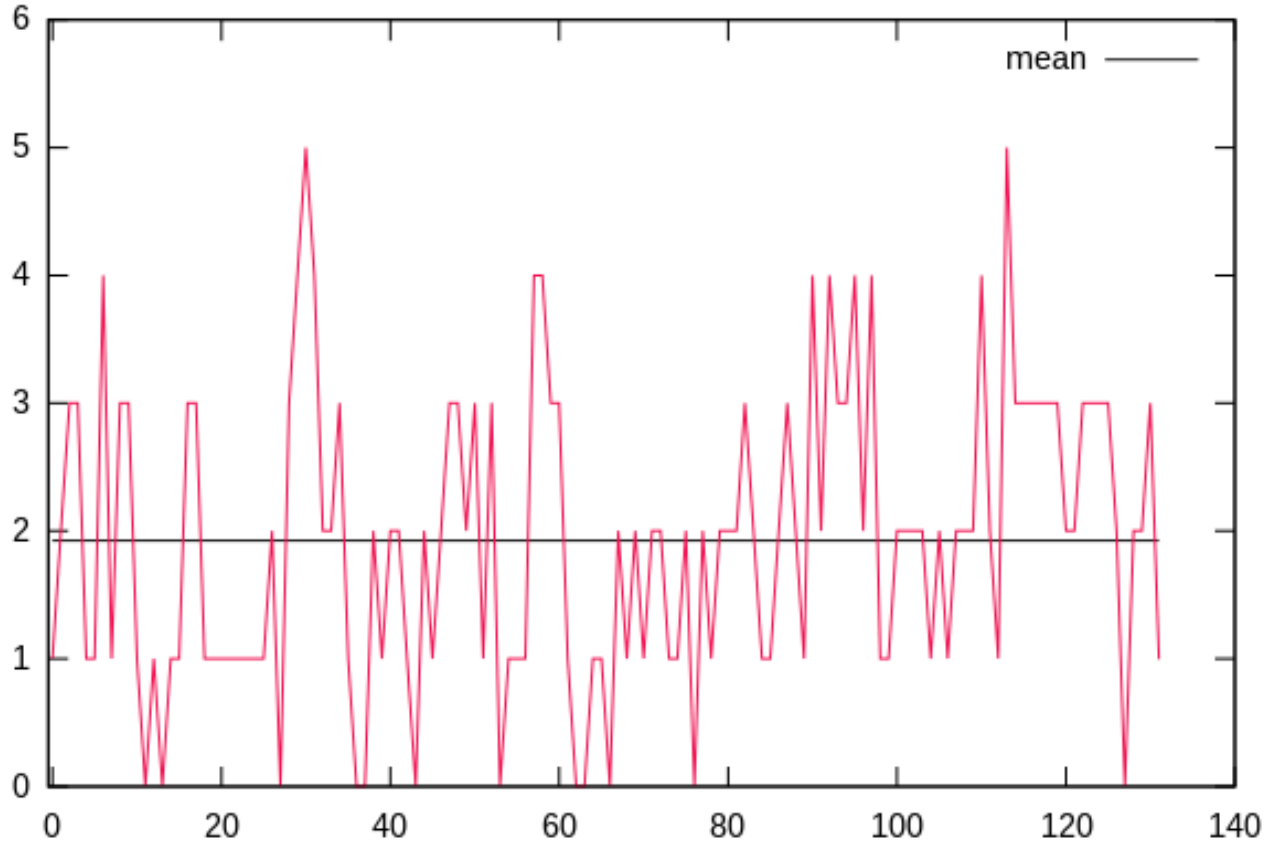
5.4.1 claims_successes_fails_per_namespace.sh

```
bashful 108 0 108
dopey 108 87 21
happy 108 108 0
sleepy 108 27 81
sneezy 108 64 44
```

5.4.2 fails_per_minute.sh

```
 1  0 1        23  22 1       45  44 2       67  66 0       89  88 2       111  110 4
 2  1 2        24  23 1       46  45 1       68  67 2       90  89 1       112  111 2
 3  2 3        25  24 1       47  46 2       69  68 1       91  90 4       113  112 1
 4  3 3        26  25 1       48  47 3       70  69 2       92  91 2       114  113 5
 5  4 1        27  26 2       49  48 3       71  70 1       93  92 4       115  114 3
 6  5 1        28  27 0       50  49 2       72  71 2       94  93 3       116  115 3
 7  6 4        29  28 3       51  50 3       73  72 2       95  94 3       117  116 3
 8  7 1        30  29 4       52  51 1       74  73 1       96  95 4       118  117 3
 9  8 3        31  30 5       53  52 3       75  74 1       97  96 2       119  118 3
10  9 3        32  31 4       54  53 0       76  75 2       98  97 4       120  119 3
11  10 1       33  32 2       55  54 1       77  76 0       99  98 1       121  120 2
12  11 0       34  33 2       56  55 1       78  77 2      100  99 1       122  121 2
13  12 1       35  34 3       57  56 1       79  78 1      101  100 2      123  122 3
14  13 0       36  35 1       58  57 4       80  79 2      102  101 2      124  123 3
15  14 1       37  36 0       59  58 4       81  80 2      103  102 2      125  124 3
16  15 1       38  37 0       60  59 3       82  81 2      104  103 2      126  125 3
17  16 3       39  38 2       61  60 3       83  82 3      105  104 1      127  126 2
18  17 3       40  39 1       62  61 1       84  83 2      106  105 2      128  127 0
19  18 1       41  40 2       63  62 0       85  84 1      107  106 1      129  128 2
20  19 1       42  41 2       64  63 0       86  85 1      108  107 2      130  129 2
21  20 1       43  42 1       65  64 1       87  86 2      109  108 2      131  130 3
22  21 1       44  43 0       66  65 1       88  87 3      110  109 2      132  131 1
```

5.4.3 fails_per_minute-graph.sh



5.4.4 Task_Duration.sh

```
 1  sneezy.67 56085
 2  sneezy.70 53042
 3  sneezy.68 34040
 4  sneezy.69 55038
 5  sneezy.66 45047
 6  sneezy.73 46089
 7  sneezy.75 58058
 8  happy.70 15035
 9  happy.67 19035
10  happy.66 17031
11  happy.69 19052
12  happy.68 18031
13  sneezy.77 34033
14  dopey.69 59040
15  dopey.68 44040
16  sneezy.83 50033
17  sneezy.82 34082
18  sneezy.85 37029
19  dopey.72 53035
20  dopey.71 36032
21  dopey.75 49031
22  dopey.74 31026
23  sneezy.89 53026
24  sneezy.86 45027
25  sneezy.88 57025
26  sneezy.87 30032
27  happy.75 19093
28  happy.72 19083
29  happy.73 18030
30  happy.74 15064
31  happy.71 16072
32  dopey.80 46030
33  dopey.79 40025
34  dopey.78 46031
35  dopey.76 57028
36  dopey.77 55026
37  dopey.83 55101
38  dopey.84 48040
39  dopey.82 46046
40  dopey.81 38032
41  dopey.88 30082
42  dopey.89 44086
43  dopey.86 44027
44  dopey.90 47024
45  dopey.87 32081
46  dopey.95 44028
47  dopey.91 53025
48  dopey.94 41024
49  dopey.93 34029
50  dopey.92 47024
51  happy.77 16029
52  happy.76 17029
53  happy.79 18025
54  happy.80 16026
55  happy.78 18071
56  dopey.98 53026
57  dopey.96 54031
58  dopey.97 55025
59  dopey.100 45033
60  dopey.99 42024
61  dopey.103 57080
62  dopey.105 31025
63  dopey.102 30028
```

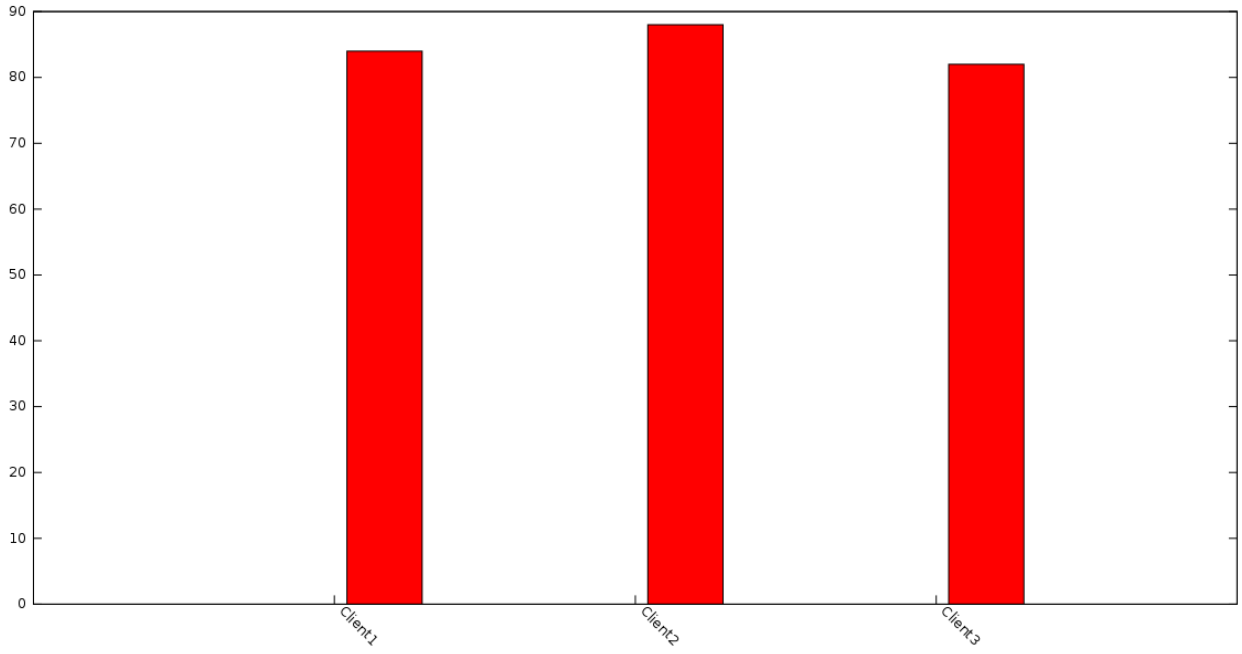
```
 64  dopey.101 34027
 65  dopey.104 46056
 66  dopey.107 38025
 67  dopey.106 55024
 68  dopey.108 38023
 69  dopey.111 34026
 70  dopey.112 56029
 71  happy.81 19030
 72  happy.82 16027
 73  happy.84 19027
 74  happy.83 15032
 75  happy.85 19038
 76  sneezy.91 33027
 77  sneezy.100 56023
 78  sneezy.99 32028
 79  sneezy.98 47024
 80  dopey.116 33024
 81  dopey.117 32029
 82  dopey.119 48028
 83  dopey.118 30031
 84  happy.87 16026
 85  happy.89 18025
 86  happy.86 15030
 87  happy.90 19024
 88  happy.88 17024
 89  happy.95 18080
 90  happy.93 18031
 91  happy.94 15024
 92  happy.92 17024
 93  happy.91 19025
 94  sleepy.74 54031
 95  sneezy.102 58023
 96  sneezy.105 58026
 97  sneezy.103 39024
 98  dopey.5 43024
 99  dopey.2 59023
100  dopey.1 50027
101  dopey.3 54027
102  dopey.4 49027
103  sleepy.79 45026
104  sleepy.76 57023
105  dopey.6 55049
106  dopey.10 41023
107  dopey.9 42024
108  dopey.8 35023
109  dopey.7 31029
110  happy.99 15025
111  happy.96 15034
112  happy.98 15024
113  happy.97 15023
114  happy.100 16024
115  sneezy.108 56028
116  sneezy.106 46025
117  happy.105 17024
118  happy.103 19029
119  happy.102 18027
120  happy.104 15025
121  happy.101 17038
122  sneezy.113 47023
123  sneezy.111 47024
124  sneezy.112 51022
125  happy.109 18079
126  happy.107 16028
127  happy.106 17023
128  happy.110 18023
129  happy.108 15025
130  sneezy.118 39024
131  sneezy.116 40023
132  sneezy.117 44024
133  happy.112 16023
134  happy.115 15023
135  happy.111 15027
136  happy.114 19026
137  happy.113 18025
138  sleepy.85 49024
139  sleepy.84 58028
140  dopey.14 41024
141  dopey.12 59024
142  dopey.11 49024
143  sneezy.4 54025
144  sneezy.3 36024
145  sneezy.2 38025
146  sleepy.86 58024
147  happy.118 18023
148  happy.117 15024
149  happy.119 15024
150  happy.116 17024
151  happy.4 18022
152  happy.2 16026
153  happy.3 15025
154  happy.5 19023
155  happy.1 16022
156  sneezy.6 53023
157  sneezy.7 53037
158  sneezy.10 59024
159  sneezy.8 52024
160  dopey.17 45024
161  dopey.19 56025
162  dopey.18 46025
163  dopey.20 43022
164  dopey.16 50026
165  sneezy.12 54024
166  sneezy.15 50023
167  sneezy.13 51026
168  happy.8 19024
169  happy.6 19025
170  happy.10 16023
171  happy.9 16025
172  happy.7 19024
173  happy.11 18026
174  happy.12 19023
175  happy.14 18023
176  happy.13 17025
177  happy.15 18023
178  sneezy.16 42022
179  sneezy.18 50025
180  sneezy.20 49028
181  sneezy.19 54024
182  sleepy.94 55024
183  dopey.21 41027
184  dopey.23 56023
185  dopey.24 36027
186  happy.19 15023
187  happy.16 18025
188  happy.17 15024
189  happy.20 15023
190  happy.18 17025
191  dopey.27 37025
192  dopey.28 52024
193  dopey.29 41026
194  dopey.26 56025
195  sleepy.103 55023
196  sleepy.102 56024
197  sleepy.105 49024
198  sneezy.21 57023
199  sneezy.23 48023
200  sneezy.25 46028
201  sneezy.24 58025
202  dopey.35 43024
203  dopey.32 36026
204  dopey.34 57025
205  dopey.31 52025
206  dopey.33 56025
207  sneezy.28 52024
208  sneezy.33 37025
209  sneezy.31 51022
210  happy.25 18022
211  happy.22 15024
212  happy.24 17026
213  happy.21 15024
214  happy.23 16025
215  sneezy.36 49024
216  sneezy.40 35025
217  sneezy.41 33022
218  sneezy.44 49064
219  sneezy.42 56024
220  sneezy.45 56025
221  dopey.37 32027
222  dopey.39 51025
223  dopey.36 55025
224  sleepy.110 46023
225  sleepy.109 58027
226  sleepy.107 51023
227  sneezy.47 53022
228  sneezy.50 50023
```

```
229  sneezy.46 48023
230  sneezy.49 56023
231  sneezy.54 48024
232  sneezy.52 48024
233  sneezy.51 42026
234  dopey.44 52079
235  dopey.42 40026
236  dopey.41 34028
237  dopey.45 36026
238  dopey.43 47068
239  sleepy.116 55023
240  happy.30 19030
241  happy.29 17026
242  happy.26 16029
243  happy.28 17027
244  happy.27 19025
245  happy.35 16040
246  happy.34 16027
247  happy.33 19026
248  happy.32 15023
249  happy.31 15025
250  dopey.47 49026
251  dopey.50 40026
252  happy.38 17027
253  happy.40 17026
254  happy.37 17026
255  happy.39 18025
256  happy.36 19026
257  happy.45 18025
258  happy.43 19076
259  happy.42 15067
260  happy.41 16042
261  happy.44 18026
262  dopey.52 39028
263  dopey.54 35025
264  dopey.53 34026
265  happy.48 15027
266  happy.46 17026
267  happy.50 19026
268  happy.47 16026
269  happy.49 18063
270  happy.51 17027
271  happy.54 19026
272  happy.53 16025
273  happy.52 18025
274  sleepy.5 54027
275  sleepy.4 54028
276  sleepy.8 57025
277  sleepy.10 50027
278  sleepy.16 53028
279  sleepy.20 47025
280  sleepy.24 55028
281  sleepy.39 56026
282  sleepy.45 50026
283  sleepy.47 49025
284  sleepy.48 47050
285  sleepy.54 50025
286  sleepy.52 57026
```

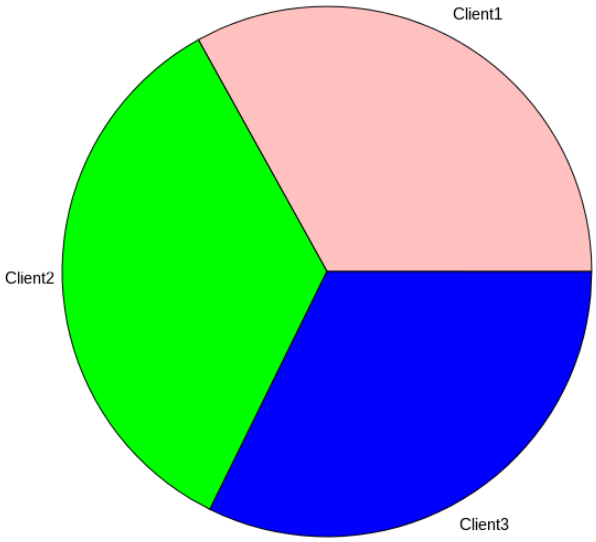
5.4.5 claims_successes_fails_per_host.sh

```
1  Client1 186 102 84
2  Client2 169 81 88
3  Client3 185 103 82
```

5.4.6 fails_per_host.svg.sh



5.4.7 fails_per_host.pie.svg.sh



5.4.8 fails_per_host.html.sh

```
<html><body>
<img src="fails_per_minute.1392043469.svg"><br />
<table border="1">
<tr>
        <td>Client1</td><td>84</td>
</tr>
<tr>
        <td>Client2</td><td>88</td>
</tr>
<tr>
        <td>Client3</td><td>82</td>
</tr>
</table><br />
<img src="fails_per_host.1392043469.svg"><br />
<img src="fails_per_host.pie.1392043469.svg"><br />
</body></html>
```

5.4.9 claims_successes_fails-global.sh

```
540 286 254
```

6 Fazit

6.1 Zusammenfassung

Wir haben im Rahmen dieser Bachelorarbeit auf Basis des Systems von Jan Dillmann ein System Entwickelt, mit dem man vergleichsweise stressfrei eine Aufgabe auf mehrere Maschienen verteilen kann.

- Es wurden unter Verwendung der API von Jan Dillmann Kommandozeilenwerkzeuge geschaffen, die die Interaktion mit dem System vereinfachen.
- Es wurde die Clientsoftware dahingehend erweitert, dass sie statt einzelnen Tasks nun Listen von Tasks bearbeitet
- Es wurde eine zentralisierte Lösung für Logging entworfen und Implementiert.
- Wir haben ein System zum Parsen und Auswerten, sowie zur Präsentation der Ergebnisse der Auswertung entworfen und implementiert.

6.2 Beobachtungen

Viele der bestehenden Systeme zum verteilten Arbeiten sind unnötig komplex und zu generisch um einen Vorteil gegenüber einer eigenen Lösung darzustellen. Zu oft wird versucht eine Lösung für jedes Problem zu bieten, statt sich auf ein Problem zu konzentrieren.

Die Verwendung einfacher, standartisierter Interfaces ermöglichte es uns in dieser Arbeit ein extrem modulares System zu schaffen, in dem jede Aufgabe von einem spezialisierten Programm erledigt wird. Durch klare und strikte Aufgabenteilung ist die Komplexität des Systems gesunken, da sich einzelne Teile problemlos ergänzen, ersetzen oder streichen lassen.

6.3 Weitere Arbeit

Das System ist bereits in einem Zustand in dem es für den Produktivbetrieb geeignet ist, doch bedarf es nochimmer einem gewissen Ausmass an Verständnis für die innere Struktur um es effektiv verwenden zu können. Dem könnte mit einem mehr nutzerzentrierten Interface begegnet werden.

Bei der Entwicklung wurden einige Annahmen getroffen, die oft nicht gegeben sind, beispielweise die Sicherheit des Netzwerkes in dem das Framework operiert. Eine Reduktion dieser Annahmen, zum Beispiel durch die Einführung von Signaturen für die Workerpakete würde es ermöglichen das System auch unter weniger kontrollierten Bedingungen zu betreiben.

Literaturverzeichnis

[1] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Jan Dillmann. Development of a task distribution framework. Bsc thesis, TU Darmstadt, 2012.

[3] R. Gerhards. The Syslog Protocol. RFC 5424 (Proposed Standard), March 2009.

[4] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.