FH | JOANNEUM
University of Applied Sciences

Implementierung eines persistenten und skalierenden Job-Schedulers

Bachelorarbeit 2
zur Erlangung des akademischen Grades
Bachelor of Science in Engineering (BSc)
eingereicht am
Fachhochschul-Studiengang **Software Design**

FH JOANNEUM (University of Applied Sciences), Kapfenberg

Betreuer: FH-Prof. Dipl.-Ing. Dr. Egon Teiniker

eingereicht von: Mario Löfler
Personenkennzahl: 1310418017

Juni 2017

Zusammenfassung

Die wiederkehrende Erledigung von Aufgaben ist in der EDV ein häufig anzutreffendes Problem. Dabei müssen die Aufgaben in etwa zu einem bestimmten Zeitpunkt in der Zukunft erledigt werden oder wiederkehrend in vorgegebenen Zeitabständen ausgeführt werden - ohne Interaktion eines Benutzers.
Im Internet der Dinge werden solche Systeme auch oft eingesetzt um Daten von Geräten zu holen die diese nicht aktiv zur Verfügung stellen können. Im Internet der Dinge kann diese Anzahl dabei rasch hoch werden, das System muss mit der wachsenden Anzahl an Geräten in der Leistungsfähigkeit mitwachsen und die erwarteten Tätigkeiten ausführen.
In dieser Arbeit wird ein solches System das sowohl funktionell als auch in der Leistung skaliert entwickelt und prototypisch eingesetzt. Einige der wichtigen Themen werden auch in der Theorie erklärt und die Umsetzung im Code erläutert. Schwerpunkt dieser Erläuterungen sind insbesondere die Problemstellungen bei der Persistenz der durchzuführenden Aufgaben und der Sicherstellung der Leistung.
Die Umsetzung wird zur Lösung einer tatsächlichen Aufgabenstellung eingesetzt. Der Theorieteil basiert auf einer Literaturrecherche.
Zielpublikum der Arbeit sind Entwickler die an Serversystemen arbeiten die wiederkehrende Aufgeben zu erledigen haben oder ähnliche Problemstellungen im Bereich der Skalierung zu erfüllen haben.

Abstract

Recurring tasks are a common challenge in software systems. Tasks either need to be scheduled to be executed at a future point in time or need to be executed in set intervals without further user interaction.
The Internet of things often uses such systems to fetch data from devices that can not actively push the data. The number of devices within the Internet of things is often large and heterogeneous, so the tasks that need to be run are diverse. The system therefore needs to be able to scale in performance and functionality.
As a part of this thesis such a system is developed put to use in a real environment. Exemplary problems like persistence and concurrency are elaborated in this thesis. The theoretical part is backed by literature research. The target audience of this theses are developers that either face the challenge of recurring tasks or are generally interested in the scaling problems.

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Bachelorarbeit selbstständig angefertigt und die mit ihr verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle aus gedruckten, ungedruckten oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für gutes wissenschaftliches Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.

Die vorliegende Originalarbeit ist in dieser Form zur Erreichung eines akademischen Grades noch keiner anderen Hochschule vorgelegt worden. Diese Arbeit wurde in gedruckter und elektronischer Form abgegeben. Ich bestätige, dass der Inhalt der digitalen Version vollständig mit dem der gedruckten Version übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Graz, 21. Juni 2017 Mario Löfler

Inhaltsverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel stelle ich das zugrundeliegende Problem einführend vor. Aus einer allgemeinen Beschreibung werden im Zusammenspiel mit einer beispielhaften Verwendung der zu entwickelnden Software die grundlegenden Anforderungen erarbeitet. Aus diesen werden die Ziele dieser Arbeit definiert.

1.1 Motivation

1.2 Beispielhafte Verwendung

asdasdsa asdasd

1.3 Anforderungen

adsasd

1.4 Ziele der Arbeit

asdasd

1.5 Verwendete Programmierumgebung

Kapitel 2

Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen zu den wichtigsten Bereichen der zu entwickelnden Software erläutert. Auf eine Einführung in die verwendete Programmiersprache und deren Ökosystem wurde aus Platzgründen verzichtet. Es werden ausgewählte Themen aus den Bereichen relationale Datenbank und Concurrency beschrieben.

2.1 Relationale Datenbanken

asda

2.1.1 Modellierung

dasds

2.1.2 Primary Key

dfsdf

2.1.3 Locks und Isolation

asdas

2.2 Concurreny

asdas

2.2.1 Threads

asdas

2.2.2 Synchronisierung

dasdas

Kapitel 3

Umsetzung

In diesem Kapitel wird die Umsetzung der Anforderungen in die Software erläutert. Es werden dabei besonders die Aspekte der Datenpersistenz, der Concurrency und der Skalierbarkeit vorgestellt.

3.1 Überblick und Architektur

asdasd

3.2 Datenbank

asdasd

3.2.1 Datenbankmodell

aasdas

3.2.2 Zugriff

sdasdaas

3.3 Skalierung

sdfsfs

3.3.1 Skalierung der Leistung

sdsds

3.3.2 Funktionale Skalierung

sdsd

```
foreach ( string file in  workerDirectory.GetFiles( "*.dll" ) )
{
        try
        {
                Assembly assembly = Assembly.LoadFile( file );
                logger.LogDebug( "[REPO]    scanning '{0}'", file );

                foreach ( Type typeToLoad in assembly.GetTypes()
                                                .Where( t => t.ImplementsInterface<IWorker>() ) )
                {
                        logger.LogDebug( "[REPO] loading '{0}'", typeToLoad.FullName );

                        IWorker w = (IWorker)Activator.CreateInstance( typeToLoad );

                        w.Initialize( ObjectRepository.Get<IConfigurationProvider>( ), logger );

                        if( ! repository.TryAdd( typeToLoad.FullName, w ) )
                        {
                                throw new InvalidOperationException
                                ( $"Could not add class '{typeToLoad.FullName}' to repository");
                        }
                }
        }
        catch ( Exception x )
        {
                logger.LogError( "[REPO] Error loading file '{0}': {1}"
                                , file
                                , x.GetDetailString( ) );
                sb.AppendLine( $"Error loading file '{file}': {x.GetDetailString()}");

        ++errorCount;
        }
}
```

Listing 3.1: Dynamic Loading

3.4 Concurrency

dasdasd

3.4.1 Threadpool

asda

3.4.2 Producer-Consumer

adadsa

3.4.3 Blocking-Queue

Listing BlockingQueue-V1

```
1           /// <summary>
2       /// own, experimental, implementation of ablocking, notifying Q for a producer/ consumer model
3       /// </summary>
4       public class MyBlockingQueueV1<T>
5       {
6               /// <summary>
7               /// The backing store used for the elements
8               /// </summary>
9               private Queue<T> backingStore = new Queue<T>();
10
11              /// <summary>
12              /// lock to protect the q
13              /// </summary>
14              private object myLock = new object();
15
16
17
18              public MyBlockingQueueV1()
19              { /* empty */ }
20
21              public void Enque( T newElement )
22              {
23                      lock ( myLock )
24                      {
25                              backingStore.Enqueue( newElement );
26                      }
27              }
28
29
30              public T Dequeue()
31              {
32                      lock ( myLock )
33                      {
34                              return backingStore.Dequeue();
35                      }
36              }
37
38              public T TryDequeue()
39              {
40                      lock( myLock )
41                      {
42                              if( !IsEmpty() )
43                              {
44                                      return Dequeue();
45                              }
46                              else
47                              {
48                                      return default( T );
49                              }
50                      }
51              }
52
53              public bool IsEmpty()
```

```
54                  {
55                          lock ( myLock )
56                          {
57                                  return backingStore.Count == 0;
58                          }
59                  }
60          }
```

3.5 Verbesserungsmöglichkeiten

Kapitel 4

Konklusio

4.1 Verwendbarkeit der erstellten Software

dsds

4.2 Abschlussbemerkung

asdasd

Abbildungsverzeichnis

Listings