

Performance-Evaluation von Open Source Reporting Engines

Bachelor-Thesis im Studiengang INF

von

Denis E. Bittante Fanacon

Eingereicht bei:

Dr. Oliver Kamin
Departement Informatik
Departementsleiter

Referent:

Heinrich Zimmermann
Prof. Dr.
Fachbereich Enterprise Computing

St. Gallen, 26. Februar 2018

Abstract

Developing software for a microservice on a Platform as a Service (PaaS) is a common use case. Finding the fitting library is essential, due to the strict resource policies on PaaS.

This work evaluates reporting engines, which are employed to create PDFs, which are Apache PDFBox, JasperReports and iText. With several prototypes on a PaaS, the libraries behaviour was observed during performance tests. The resources were used at best by Apache PDFBox with respect to CPU and Memory. JasperReports showed a great need for memory and CPU. The iText library has shown a balanced use of the memory and CPU. Throughput and latencies where the most performant with Apache PDFBox followed by iText and JasperReports.

The results showed that out of the three evaluated open source reporting engine, Apache PDFBox and iText were eligible to be used in combination with microservices. We also saw that this results can fluctuate depending on how the tests are defined.

Zusammenfassung

Die Enterprise Software ist im Wandel und Microservices werden immer häufiger eingesetzt. Diese Arbeit widmet sich der Erforschung der geeigneten Open Source Reporting Engine (OSRE) im Hinblick auf mögliche Ursachen für Performance-Engpässe und deren Einsatz im Umfeld von Containern und Infrastructure as a Service (IaaS).

Die OSRE wurden in Prototypen genutzt, um verschiedene PDFs zu erstellen. Die Prototypen wurden als REST-Services auf IaaS eingesetzt und gegen einen Lasttest laufen gelassen. Die Tests gaben Aufschluss über Latenzzeit, Memory-Verbrauch und Verfügbarkeit des Services.

Die Ergebnisse haben gezeigt, dass iText als performantes Tool mit gutem API und guter Nutzung der Ressourcen ist. Reine Performance wird mit Apache PDFBox besser erreicht, da dieses Tool auf Performance getrimmt wurde. Hierbei fehlen aber die Mittel, um schöne oder komplexe Layouts zu generieren. Ein maximaler Durchsatz konnte mit Apache PDFBox

bei 151 Anfragen pro Sekunde verzeichnet werden. JasperReport hat sehr gut abgeschnitten im Layout und im Usability-Teil, aber ist kein Kandidat für die performante Umsetzung von Services. Über alle Tests hinweg hat JasperReport die niedrigste Performance erreicht. Die Experimente deuten darauf hin, dass die Auswahl eines geeigneten Frameworks eine wichtige Rolle spielt, wenn eine hohe Performance erreicht werden soll.

Inhaltsverzeichnis

1 Einleitung	1
2 Grundlagen	2
2.1 Performance	2
2.1.1 Metriken Performance-Messungen	2
2.1.2 Metriken - Ressourcen	3
2.1.3 Plattform as a Service	3
2.2 Open Source Reporting Engine	4
2.2.1 JasperReports®	4
2.2.2 iText	4
2.2.3 Apache PDFBox	5
2.2.4 Gegenüberstellung	5
2.3 Abgrenzung der Arbeit	5
2.3.1 PDF-Implementation und Layout	6
2.3.2 PaaS	6
3 Prototyp	7
3.1 Schnittstelle	7
3.1.1 Requests	8
3.1.2 Responses	10
3.2 PDF Generierung	10
3.2.1 Build / Deployment	11
3.2.2 Technische Details	12
4 Evaluation	14
4.1 Vorbereitung	14
4.1.1 Last-Strategie	14
4.1.2 Ramp-Up	15
4.1.3 Last	16
4.2 Test Durchführung	16
4.2.1 JMeter	16
4.2.2 Test Szenarien	17

Inhaltsverzeichnis

4.3 Konsolidierung	20
4.3.1 Log Datei	20
4.3.2 JMeter Testresultate	21
5 Resultate	22
5.1 Performance	22
5.1.1 Durchsatz nach Anfragen	22
5.1.2 Durchsatz nach Bytes	23
5.1.3 Latency	24
5.1.4 Verfügbarkeit	25
5.2 Ressourcen	27
5.2.1 Arbeitsspeicher	27
5.2.2 Prozessor Last	28
5.3 Reporting Engines - Verhalten	29
5.3.1 Requestgrösse	29
5.3.2 Virtuelle User	30
5.4 Ursachenanalyse	32
6 Fazit	33
6.1 Implementation	33
6.2 Performance	34
6.2.1 Throughput	34
6.2.2 Latency	34
6.2.3 Verfügbarkeit	34
6.3 Ressources	34
6.3.1 RAM	34
6.3.2 CPU	35
6.4 Wer hat nun die beste Performance?	35
6.4.1 Apache PDFBox	36
6.4.2 iText	36
6.4.3 JasperReports	36
6.5 Ausblick	36
Abbildungsverzeichnis	39
Literaturverzeichnis	40

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Evaluation von OSRE. Die Frage nach der leistungsfähigsten OSRE ist von besonderem Interesse, weil immer häufiger SOA im Einsatz sind. Diese Architekturen befassen sich mit verteilten Systemen, die sich gezielt auf eine Aufgabe fokussieren. Dies ist in dieser Arbeit das Generieren von Reports im PDF-Format.

Die Evaluation von OSRE ist von Interesse, da sich die reine Evaluation der Performance auf die Infrastruktur auswirkt. Je effizienter und schneller die OSRE die Anforderungen erfüllen, desto schwächer bzw. günstiger kann die Infrastruktur ausfallen, was auch bedeutet, dass diese wirtschaftlich gesehen schneller rentieren werden. Im aktuellen Markt zeigt sich der Trend zu Containern und Platform as a Service (PaaS), welche horizontal skaliert sind und wo aus Sicht der Hardware keine Grenzen gesetzt sind. Performance- und Lasttests können in diesem Anwendungsfall (Generieren der PDFs) aufzeigen, wie sich ein solcher Webserver verhalten würde. Diese Services können unter Umständen Ausfälle und Leistungsengpässe einzelner Container oder Knoten im Verbund vorhersagen. Damit lassen sich Infrastrukturen planen und administrieren, um diese z.B. in Bezug auf die aktuelle User-Aktivität auszulegen, auch wenn diese auf einer PaaS oder Infrastructure as a Service (IaaS) betrieben werden sollen.

2 Grundlagen

Welche Frameworks in dieser Arbeit mittels welcher Methoden und Metriken analysiert werden.

2.1 Performance

Ian Molyneaux definiert eine performante Anwendung wie folgt:

“Eine gut funktionierende Anwendung ist eine, die es dem Endbenutzer ermöglicht, eine bestimmte Aufgabe ohne übermässige Verzögerung oder Irritation auszuführen.”[1]

Die Schlussfolgerung davon ist, dass jeder Performance verschieden empfinden kann, aber auch, dass für jede Aufgabe das geeignete Mittel gewählt wird, das zu keiner oder nur geringer Verzögerung führt. Eine Verzögerung kann situativ verschieden empfunden werden.

Performance ist dennoch nur ein Qualitätsmerkmal von vielen, das gemessen werden kann. Als nicht funktionale Anforderung werden neben Performanz- auch Speicherplatzanforderungen oder Benutzbarkeitsanforderungen genannt [2, S. 120]. Jede dieser Anforderungen kann mit der geeigneten Methodik erhoben werden. Es können als Beispiel Lasttest und Performance-Messungen auf ein bestehendes System, falls ein solches existiert, durchgeführt werden, um zu erkennen, ob eine Applikation viel oder wenig Speicher verbraucht und somit der Speicherplatzanforderung gerecht wird. Auch die Benutzbarkeit kann mittels Interaktionstests zwischen dem User und der Applikation evaluiert werden. Dies kann zeigen, ob der User auf Anhieb zurechtkommt oder das Navigationskonzept überdacht werden soll. Um die Performanzanforderungen zu analysieren, können Prototypen gebaut werden, die mittels Performancemessungen auf verschiedene Metriken hin analysiert werden. Diese Vorgehensweise wurde für diese Arbeit gewählt.

Die Performance kann mittels verschiedener Metriken ausgedrückt werden. Folgende Metriken sind für diese Arbeit relevant:

2.1.1 Metriken Performance-Messungen

Latenzzeit (clientseitig): Ist die Zeit ab dem Senden der Abfrage, bis sie den Zielort erreicht hat und wieder zurück beim Client ist. Sie beinhaltet die Latenzzeit des Servers und des Netzwerkes.

Reaktionszeit (Response Time): Die Antwort-Zeit ist die Zeit, die benötigt wird von der Abfrage, bis alle Antworten am System empfangen werden. Das beinhaltet die Latenzzeit des Netzwerks, des Servers und eine allfällige Latenzzeit des Quellsystems.

Durchsatz (Throughput): Der Durchsatz ist die Anzahl der zu verarbeitenden Abfragen oder Bytes, die in einer definierten Zeiteinheit erledigt wurden. Es werden meist viele Stichproben gemacht, um den Verlauf protokollieren zu können. Der gesamte Durchsatz wird demzufolge so berechnet:

$$\text{Durchsatz} = \frac{\text{anz. Abfragen}}{\text{Gesamtzeit}} \quad (2.1)$$

Verfügbarkeit (Availability): Die Verfügbarkeit ist so lange gewährleistet, bis der User des Service die Antwort bekommt. Sobald die Antworten des Servers zu stark verzögert oder korrupt sind, ist der Service nicht mehr verfügbar.

2.1.2 Metriken - Ressourcen

Damit die clientseitigen Performance-Messungen nachvollziehbar werden, wurde entschieden, auf der Service-Seite einige Performance-Indikatoren zu protokollieren. Diese sollen dann mit der guten oder schlechten Performance in Zusammenhang gebracht werden. Dabei sind diese Metriken für diese Arbeit gewählt worden.

Memory RSS: Das RSS steht für 'resident set size', was die Grösse der Prozesse bezeichnet, die im Hauptspeicher gehalten werden. Diese Messung wird in MB ausgedrückt und wird jede Minute evaluiert. Das RSS wird über alle Dynos hinweg ausgegeben.

Memory SWAP: Der SWAP wird dann eingesetzt, wenn die Prozesse keinen verfügbaren Speicherplatz im RSS finden. Die Prozesse werden somit auf der Festplatte sistiert. Dabei sind die Einstellungen des unterliegenden Systems ausschlaggebend, wenn dies passiert.

Web-Server - CPU (Dyno Last) Das ist die Anzahl der Prozesse, die auf eine Ausführung warten und sich im Status 'Ready' befinden. Dabei werden die Stichproben über eine und fünf Minuten aufgezeichnet.

2.1.3 Plattform as a Service

Die Performance-Messungen auf einer PaaS auszuführen, bringt verschiedene Vorteile, wie z.B. die Möglichkeit, eine relativ grosse Infrastruktur für eine kurze Zeit günstig zu mieten. Ebenfalls kann diese relativ schnell an die Bedürfnisse angepasst werden, also z.B. die Anzahl Instanzen dynamisch anzupassen und somit auf die Bedürfnisse eines Anwendungsfalls reagieren zu können. Ebenfalls kann von den voreingestellten Konfigurationen profitiert werden, da diese für die meisten Fälle bereits optimal definiert wurden. Während eines Performancetests ist es auch nötig, die Ressourcen zu protokollieren, was auf einer

2 Grundlagen

PaaS dank dem Einsatz von Ad-Ons leicht zu erreichen ist. Doch kann eine PaaS auch Nachteile mit sich bringen. Beispielsweise kann diese nicht immer ganz zuverlässig sein, ein Server kann vielleicht nicht wie gewohnt gestartet werden oder bricht abrupt ab. Dies stellt meist keine weitere Probleme, da diese Instanzen leicht wieder gestartet oder gelöscht werden können.[1, Kap. 3]

Heroku, die gewählte PaaS, bietet eine Möglichkeit. In diesem Fall wurde ein sogenannter Dyno Typ 'Standard-2x' angewendet. Dieser Typ von Dyno verfügt über 1024 MB RAM und kann bis hundertfach horizontal skaliert werden (Scaling out). Der Dyno verfällt nicht in den Schlafmodus, was bedeutet, dass die ersten Anfragen ebenfalls nicht länger dauern als die anschliessenden.

2.2 Open Source Reporting Engine

Als Open Source Reporting Engine werden die Libraries definiert, welche als Einsatzgebiet das Rendern und Generieren von Reports haben. Ein Einsatzgebiet kann das Generieren von Flugtickets, das grafische Aufbereiten von Geschäftszahlen oder einfach nur das Darstellen von Fliesstext sein.

Es gibt im Markt verschiedene dieser OSREs wie Apache PdfBox , iText ,BIRT, JasperReports, um nur einige der Engines zu nennen. Viele dieser OSREs sind auf ein Einsatzgebiet spezialisiert. Meist können diese OSREs nicht nur Reports generieren, sondern diese auch verarbeiten oder verändern [3, Kap. 10]. Als Zielformat wird oft PDF benutzt. Dieses Format ist ideal, um Druckvorlagen zu erstellen oder die Informationen auf verschiedenen Endgeräten darzustellen. Das macht PDF zum robusten und versatilen Dateiformat. Wir wollen im Folgenden die für die Prototypen genutzten OSREs näher betrachten.

2.2.1 JasperReports®

Das von Teodor Daciuc lancierte Reporting Tool wurde aufgrund grosser Nachfrage im November 2001 in der Version 0.1.5 lanciert [4, Kap. 1]. Die Reports werden als Vorlagen im XML-Format definiert. Diese Vorlagen können für die Verarbeitung vorkompiliert werden. Der Visual Designer iReport macht es möglich, das Layout grafisch zu bearbeiten. Dank diesem Ansatz kann das Layout vom Programmcode getrennt werden. JasperReports nutzt als Framework iText um einige der Features umzusetzen.

2.2.2 iText

Als die Studenten- und Kursverwaltung einer Universität fertiggestellt wurde, musste eine geeignete Druckmöglichkeit gefunden werden. HTML und Word wurden als ungeeignetes

2.3 Abgrenzung der Arbeit

Mittel angesehen, und man entschied sich, PDF einzusetzen. Da im Jahr 1998 noch keine geeignete FOSS für PDF existierte, wurde eine eigene Library geschrieben, die heute als iText bekannt ist. iText wird heute in verschiedenen Softwares eingesetzt und kann in Tools wie Eclipse/BIRT und JasperReports gefunden werden. NASA nutzt die Library für ihre Blueprints, und Google Calendar gehört ebenfalls zu den Nutzern dieser Library [5, Kap. 1].

2.2.3 Apache PDFBox

Im Jahr 2002 wurde für das Projekt Apache Lucene eine Library gesucht, um den Inhalt von PDFs zu extrahieren. Das Ziel war, den Inhalt durch Lucene zu indexieren. Diese Library wurde von Ben Litchfield initialisiert, und seit der ersten Version wurden verschiedene Performance-Verbesserungen eingeführt. Lucene wird von Wikipedia und Twitter genutzt. Zu den Features von Apache PDFBox gehört nicht nur das Extrahieren von Inhalten, sondern auch das Teilen, Vereinen und Füllen von PDFs [6].

2.2.4 Gegenüberstellung

Die hier vorgestellten Open Source Frameworks können mehr als nur PDFs erstellen, obwohl einzig diese die Eigenschaft haben, die in dieser Arbeit geprüft wird. Die meisten dieser Frameworks können Inhalte aus bestehenden PDFs entnehmen, PDFs teilen oder vereinen. Wie die Tabelle 2.1 zeigt, sind bei der Ausbaustufe trotzdem Unterschiede festzustellen.

	Apache PDF Box	iText	JasperReport
PDF-Metadaten	X	X	X
PDF generieren	X	X	X
Bilder einfügen	X	X	X
Formen zeichnen	X	X	X
Tabellen einfügen		X	X
Tabellen-Zellen formatieren		X	X
Layout-Vorlage definieren			X

Tabelle 2.1: OSRE PDF-Erstellen Features

2.3 Abgrenzung der Arbeit

Es ist in dieser Arbeit nicht möglich alle Aspekte von Performance und deren Analyse vorzunehmen. Darum werden die hier erarbeiteten Erkenntnisse nicht als Allgemein gültig definiert.

2 Grundlagen

2.3.1 PDF-Implementation und Layout

Die generierten PDFs werden anhand des gleichen Inputs generiert, die Verarbeitung dieser Inputs wird jedoch von jeder Reporting Engine verschieden aufbereitet. Aufgrund des Funktionsumfangs der OSRE kann nicht gewährleistet werden, dass die leistungsstärkste Umsetzung gewählt wurde oder die idealen Features genutzt wurden. Da die ORSE verschiedene Strategien besitzen, wie PDFs erstellt werden, und nicht alle APIs gleich ausgereift sind, sind einige der Layouts nicht vollständig von einer OSRE auf die andere übertragbar. Darum wurde Wert darauf gelegt, dass die Reporte sich schlussendlich ähneln und die Informationen angezeigt werden können.

2.3.2 PaaS

Die verschiedenen Applikationen wurden auf der PaaS Heroku betrieben und getestet. Da die Performance einer Applikation durch verschiedene nicht beeinflussbare Ereignisse verändert wird, wie z.B. die Netzwerklatenzzeit oder die Last selbst auf den Providern, ist zu erwarten, dass die Ergebnisse immer unterschiedlich sein werden. Ein Aspekt, der in dieser Arbeit nicht betrachtet wird, der jedoch im Zusammenhang mit Performance-Tests auf einer PaaS sehr relevant ist, ist das Load-Balancing. Einerseits werden die User, die die Anfragen stellen, immer die gleiche IP-Adresse besitzen. Auch werden keine Autoscaling-Dynos gestartet, was bedeutet, dass die Anfragen nicht auf mehrere Dynos aufgeteilt, sondern nur auf einem Server verarbeitet werden. Es ist somit klar, dass diese Test-Umgebung mit einer produktiven Umgebung nicht vergleichbar ist und somit die Ergebnisse nicht absolut übertragbar sind.

3 Prototyp

Um herauszufinden, welche der OSRE am leistungsstärksten ist, wurden drei Prototypen erstellt. Mit diesen soll erforscht werden, wie sich eine Applikation beim Einsatz der verschiedenen ORSE verhält. Es wurde darauf geachtet, dass der Prototyp nur genau die Komponenten besitzt, die für die Aufgabe benötigt werden.

Jeder Prototyp ist zweiteilig. Die Prototypen besitzen die gleiche REST-Schnittstelle, welche die Anfragen entgegennimmt. Die REST-Schnittstelle leitet diese Anfrage an die entsprechenden Implementationen weiter. Dort werden diese Anfragen in konkrete PDFs umgesetzt. Im Folgenden werden die grundlegenden Aspekte der Implementationen näher betrachtet.



Prototyp
Der Prototyp kann unter <https://github.com/denisbittante/dinf/tree/master/loadtest/testdata> gefunden werden.

3.1 Schnittstelle

Die Schnittstelle wurde so implementiert, dass eine HTTP-POST-Anfrage entgegengenommen werden kann. Der Request-Body wird dem Interface 'PdfEngine' weitergegeben. Dieses Interface wurde mithilfe der Spring-Annotation @Autowired eingesetzt. Damit besteht eine lose Koppelung der PDF-Implementation und der Schnittstelle gegen aussen. Dank dieser Aufstellung kann nun einfach eine beliebige Implementation aufgerufen werden.

Die REST-Schnittstelle bietet die drei PDF-Szenarien über die Endpunkte /scenario1, /scenario2 und /scenario3 an.

```
1  @RestController("/")
2  public class PdfServiceController {
3
4      @Autowired
5      private PdfEngine engine;
6
7      @RequestMapping(method = RequestMethod.POST, path = "scenario1")
```

3 Prototyp

```
8  public @ResponseBody PdfResponse scenario1(@RequestBody PdfRequestScenario1 req)
9  {
10    return engine.createPdfSzenario1(req);
11  }
12 }
```

3.1.1 Requests

Die Endpunkte werden mit den folgenden POST-Bodys angesprochen.

Szenario 1

Das für das erste Szenario bildet das Modell für eine Aktivität ab. Diese soll einen Titel und eine Zeitangabe besitzen, wann diese stattfindet. Es wurden verschiedene Informationen hinterlegt, die für eine Freizeitbeschäftigung interessant wären, wie: Wer organisiert und welche Freunde oder Helfer werden eingeladen? Wo soll es stattfinden?

```
1 {"data": [
2   {
3     "title": "Sommer-Fest",
4     "datumVon": "12.02.2017 16:00",
5     "datumBis": "12.02.2017 23:00",
6     "description": "Bringt Essen und gute Laune mit",
7     "place": "Musterstrasse 12\n888 Musterstadt\nSchweiz",
8     "incharge": "Denis Bittante",
9     "helper": "Lars Hauser, Max Muster",
10    "author": "Denis Bittante"
11  }, ...
12 ]}
```

Szenario 2

Szenario 2 ist sozusagen eine Zusammenfassung oder das Programm der interessanten Aktivitäten an einem Tag anstehen werden. Die Daten fassen die Eckdaten einer Aktivität zusammen wie der Organisator und die Zeitangaben. Es gibt keine Details aber eine Möglichkeit eine Fussnote zu versehen die nach jedem Tag erscheinen soll.

```
1 {"group": [
2   {
3     "title": "21.2.12 Vortraege fuer diesen Tag",
4     "entries": [ {
```

```

5      "time": "19:00",
6      "title": "Ein Titel",
7      "person": "D.Bittante",
8      "isSubtitle": false
9    },
10   ...
11   "footer": "Special Information as Footnote"
12 },
13 ]

```

Szenario 3

Im dritten Szenario wurde entscheiden, eine Kontaktliste zu erstellen, damit die Helfer und Freunde auch ausgedruckt werden können. Im Hinblick darauf, wie die Bilder und Tabellen eingefügt werden können, wurde hier entschieden, die Personendaten wie Vorname, Familiennname und Adress- und Kontaktdaten zu hinterlegen, aber auch die Information zum Geschlecht. Dabei steht 'm' für männlich (male) und 'f' für female. Die Gender-Information soll dann auf der Kontaktliste als Gender-Symbol dargestellt werden.

```

1 {"contacts": [
2   {
3     "gender": "m",
4     "name": "Musternname",
5     "firstname": "Max",
6     "address": "Musterstrasse 12",
7     "zip": "9000",
8     "place": "Ort",
9     "tel": "+41 55555 55 55",
10    "mob": "+41 55555 55 55",
11    "birthday": "2.2.1988",
12    "note": "schlaue Notiz fuer diese Person",
13    "email": "max.muster2000@gmx.ch"
14  },
15 ]
}

```

JSON Request



Vollständige Request können unter <https://github.com/denisbittante/dinf/tree/master/loadtest/testdata> gefunden werden.

3 Prototyp

3.1.2 Responses

Die Responses der REST-Schnittstelle wurden einfach gehalten, daher werden die OSRE und das Szenario angegeben und im Feld Status, ob die Verarbeitung korrekt abgeschlossen wurde. Das generierte PDF ist als base64 encodiert und als Feld file in der Response zu finden.

```
1 {  
2   "name": "ApachePdfBox",  
3   "description": "Szenario 1",  
4   "file": "JVBERi0xLjQKJfb ... hyZWyKMTc2NAo1JUVPRgo=",  
5   "status": "ok"  
6 }
```

3.2 PDF Generierung

Beim Einsetzen des Interfaces PDFEngine werden je nach Build eine der drei Implementierungen instanziert (siehe Abbildung 3.1). Jede dieser Klassen hat alle drei Szenarien zu implementieren.

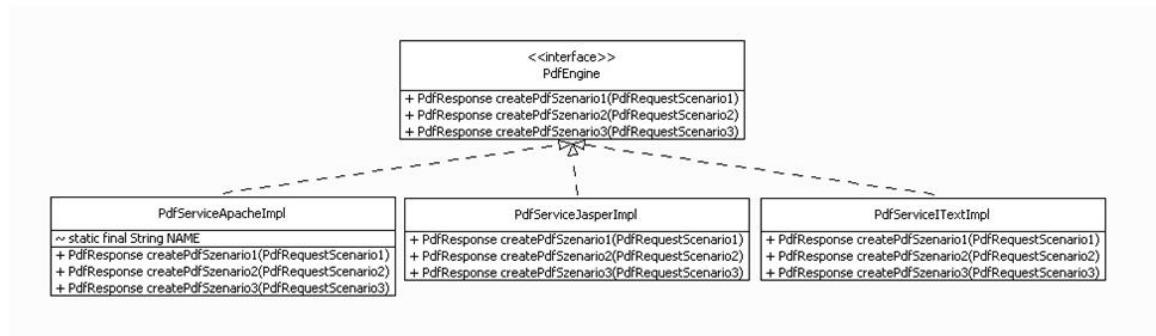


Abbildung 3.1: UML PDFEngine Implementierungen

Die einzelnen Szenarien wurden ebenfalls als Interfaces deklariert, damit jedes dieser Szenarios nicht anders oder nur teilweise implementiert wird. Als Basis-Klasse wurde die AbstractScenario-Klasse implementiert. Diese regelt den Aufbau, wie die temporären Files auf dem System abgelegt werden und wie diese wieder abgeräumt werden. Diese abstrakte Klasse übernimmt ebenfalls den Aufbau und das Handling des PdfRequest als Business-Model. Dieses Model steht jeder Implementation über die Methode getModel zur Verfügung.

3.2 PDF Generierung

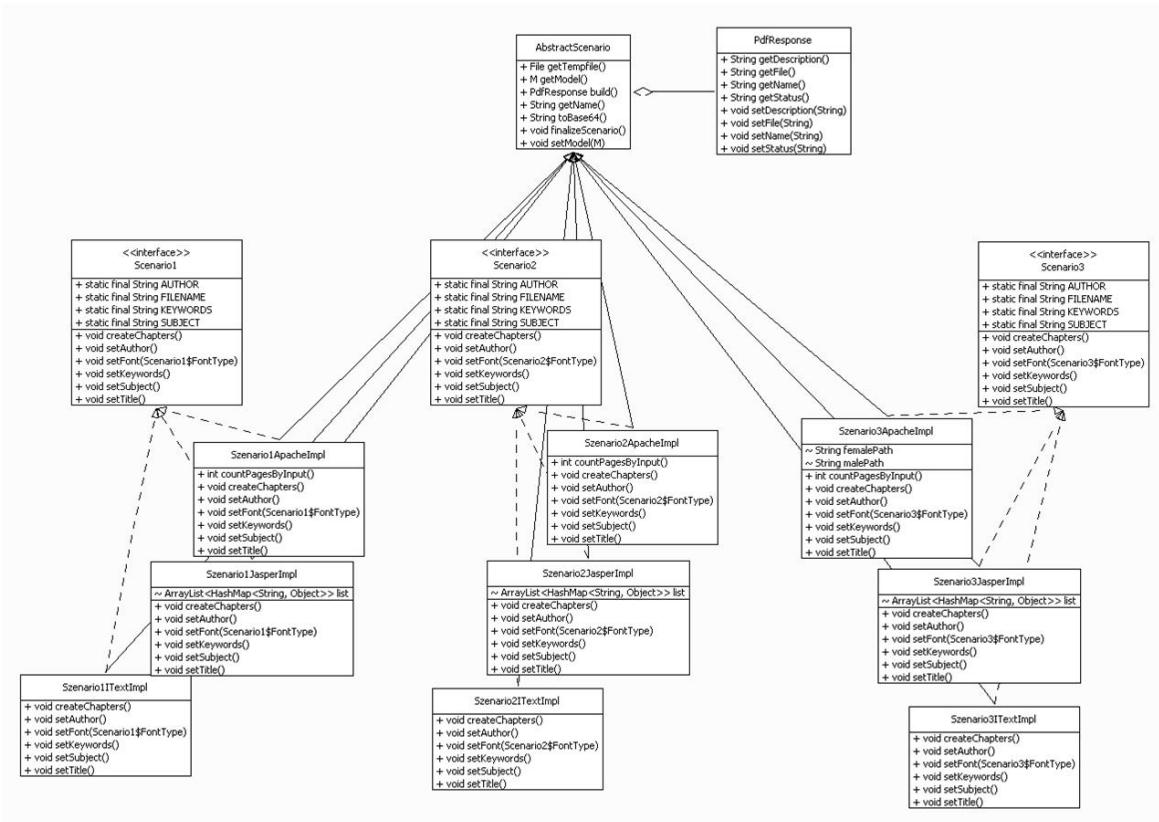


Abbildung 3.2: UML Szenario Implementierungen

3.2.1 Build / Deployment

Die einzelnen Jars mit den gewünschten Implementierungen können mittels Maven gebildet werden. Dazu wurde für jedes Produkt ein Profil eingerichtet. Dafür muss das Projekt vollständig gebildet werden, damit die einzelnen Implementierungen als Archiv im Maven Repository erreichbar sind, und anschliessend kann der REST-Service gebildet und gestartet werden.

```

1
2
3 REM Bildet alle Maven Module
4 cd ../dinf/app/
5 mvn clean install
6
7 REM Bildet den RestService mit gewünschter OSRE Implementation
8 cd ../dinf/app/rest-springboot/
9
10 mvn clean install -Pitext

```

3 Prototyp

```
11 REM (oder)
12 mvn clean install -Pjasper
13 REM (oder)
14 mvn clean install -Ppdfbox
```

Das generierte Jar kann über das heroku-Komamndo ebenfalls über die Konsole deployed werden. Da der Stand des gewünschten Dynos meist nicht bekannt ist, wurde nach dem Deployment auch der Dyno neu gestartet.

```
1
2 REM Deployed hier denn service mit der pdfbox implementation auf den Dyno 'dinf-app'
3 heroku deploy:jar target\rest-service-pdfbox.jar --app dinf-app
4
5 REM Startet den Dyno namens 'dinf-app' erneut
6 heroku restart --app dinf-app
```

3.2.2 Technische Details

Folgende technischen Eckdaten sind in diesem Experiment von Vorteil, um ähnliche Performance-Ergebnisse erzielen zu können.

Es wurden folgende Versionen genutzt:

Software	Version
Java	jdk 1.8.0.112
Spring Boot	1.5.8.RELEASE
iText	7.1.0
Apache PDFBox - Core	2.0.1
Apache PDFBox - fontbox	2.0.0
Apache PDFBox - jempbox	1.8.11
Apache PDFBox - xmpbox	2.0.0
Apache PDFBox - preflight	2.0.0
Apache PDFBox - pdfbox-tools	2.0.0
JasperReport	6.4.3
JasperReport - fonts	4.0.0
Maven	3.3.9

Heroku (PaaS)	
Region	USA
Stack	heroku-16
Buildpacks	heroku/jvm
Dyno Type	Standard-2X
RAM	1GB
Anzahl Dyno	1

Tabelle 3.1: Technische Angaben

4 Evaluation

Ob ein Framework oder ein OSRE performant ist, kann kaum der Literatur entnommen werden. Auch Benchmarks stoßen dabei an ihre Grenzen, wenn es darum geht, Vergleiche zwischen Frameworks zu machen, die je nach Einsatzgebiet verschiedene Stärken oder Schwächen aufweisen können. Dadurch kann eine objektive Aussage zwar wahr sein, sie ist womöglich aber für das eigene Projekt nicht aussreichend oder sogar falsch. Es wurde entschieden, einen eigenen Prototypen zu erstellen, den es in dieser Arbeit zu evaluieren gilt.

Für die Evaluation wurde bestimmt, die Prototypen abwechslungsweise auf die PaaS einzusetzen, um diese separat voneinander analysieren zu können. Gegen die eingesetzten Prototypen wurden automatische Lasttest durchgeführt. Da die Performance-Messungen mehrmals zu erheben waren, wurde vorgängig ein Testplan erstellt. Damit konnten die Tests vorgängig konfiguriert und automatisiert durchgeführt werden. Die wichtigsten Performance-Messungen wurden einerseits mit dem Testtool JMeter generiert und andererseits mithilfe der Logs auf der PaaS erhoben. Dank der Konsolidierung dieser Dateien konnten Rückschlüsse auf die Performance gezogen werden.

Die Evaluation gliedert sich in drei Teile (siehe Abbildung 4.1). Diese Teile werden in den folgenden Kapiteln näher erläutert.

4.1 Vorbereitung

In der Phase Vorbereitung geht es darum, die Prototypen zu erstellen sowie den Testplan zu definieren. Im Testplan müssen verschiedene Parameter definiert werden. Wie diese gewählt werden, bestimmt der Anwendungsfall, der hier behandelt wird.

4.1.1 Last-Strategie

Eine Nutzergruppe kann auf ein produktives System verschiedene Last-Muster generieren. Dabei gelingt es den Usern, durch den Ansturm auch grosse Rechenzentren lahmzulegen. Ob alle User gleichzeitig wie z.B. beim Vorverkauf von Konzertkarten beim gebotenen Service anmelden oder gestaffelt, ist für ein System durchaus wichtig, denn je nach Service

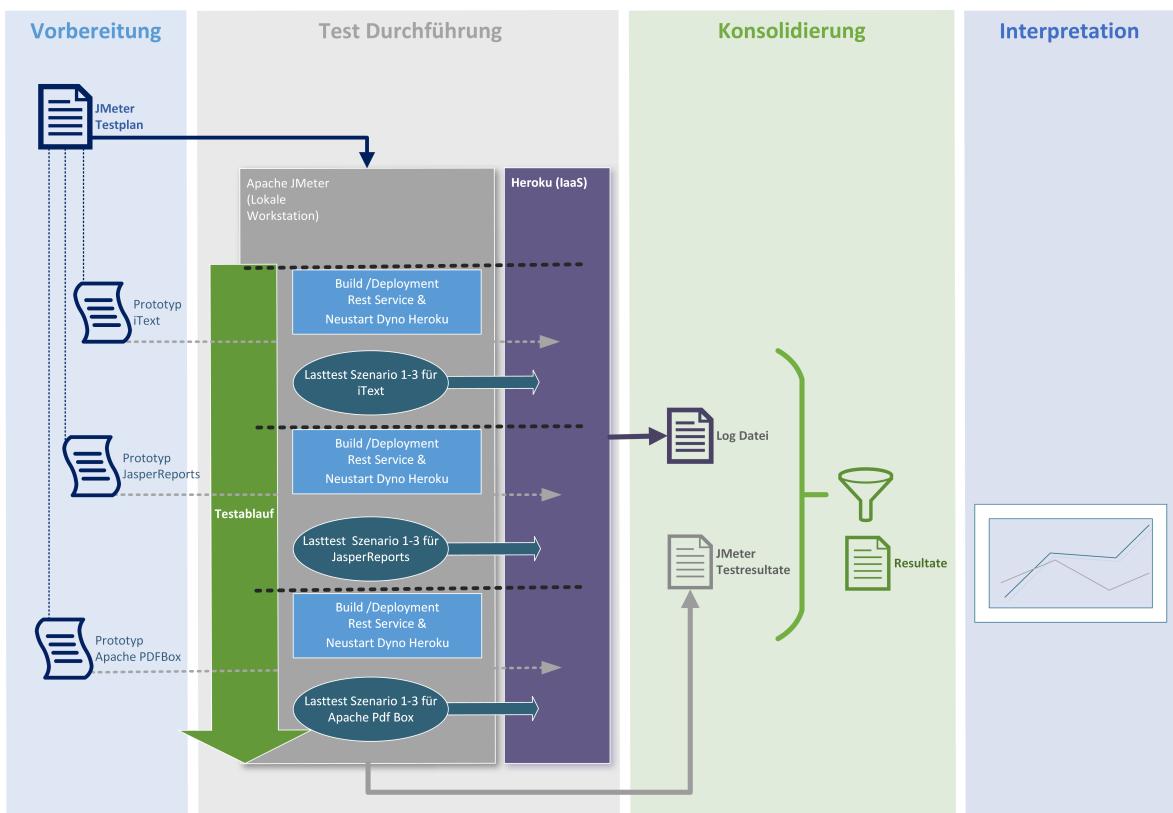


Abbildung 4.1: Evaluationsablauf

kann dieser sich mit der Anzahl Anfragen den neuen Umständen anpassen und horizontal skalieren, also die Last auf viele verschiedene Server verteilen.

4.1.2 Ramp-Up

Bei den durchgeföhrten Lasttest wurde die Infrastruktur nicht skaliert, dennoch wurde eine Ramp-Up von 10 Sekunden definiert, was bedeutet, dass die gesamte Menge der virtuellen User nach 10 Sekunden auf dem System aktiv Last erzeugen. Dieses Szenario wurde gewählt, da in der Praxis fast nie Systeme gleich vollständig ausgelastet werden.

Da es in diesen Experimenten darum geht, verschiedene Anwendungen und deren Frameworks zu evaluieren, wurde eine fixe Anzahl User definiert, die sich über 10 Sekunden hinweg alle zuschalten und 10 Minuten lang auf dem System 'arbeiten'. Somit variieren diese über den Test hinweg nicht mehr. Diese Strategie wurde gewählt, da es einfacher ist, die Messungen zu evaluieren, und die Performance-Messungen zuverlässiger gesammelt werden können.

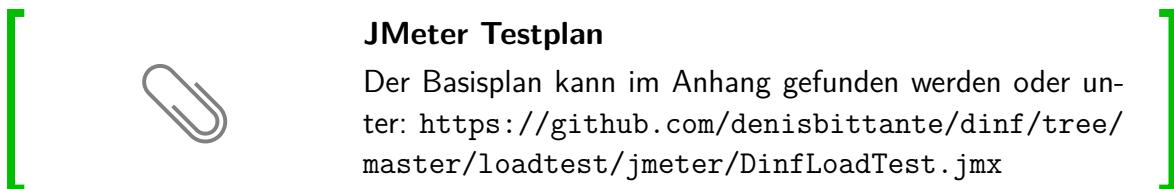
4 Evaluation

4.1.3 Last

Ein Service kann auf verschiedene Arten belastet werden. Einerseits, was öfter vorkommt, ist es das Ansteigen der aktiven User auf dem System. Diese Last wird meist über einen Scale out der Server behoben. Was ebenfalls vorkommt, ist, dass die Last auf dem Service durch grössere Anfragen gefordert wird, d.h. dass Requestgrößen ansteigen. Das kann z.B bei Services, die eine Bestandeslieferung von Einwohnern entgegennehmen, der Fall sein. Diese Anfrage nimmt oft jährlich zu, oder es ist auch möglich, dass grössere Gemeinden als Kunden dazugezählt werden dürfen.

In dieser Arbeit wurden die beiden Lasten im Testplan miteinbezogen. Bei den Tests wurden einerseits die virtuellen User von 20 auf 50 gesetzt, und andererseits wurden die virtuellen User bei 20 belassen, aber die Requestgröße verdreifacht. D.h. die zu verarbeitenden Datensätze wurden verdreifacht, nicht die Bytegröße der Anfragen. Diese Tests sollen Aufschluss darüber geben, wie sich die OSREs bei Lasterhöhung verhalten.

Die definierten Parameter und Testpläne wurden als jmx-Datei festgehalten und für die Testdurchführung aufbereitet.



4.2 Test Durchführung

Um die Performance einzelner Implementationen testen zu können, wurde eine Serie von Tests erstellt. Die Performancetests spiegeln die Anforderungen an die Services wieder. Dabei wurden verschiedene Parameter verändert, die in der Praxis ebenfalls anzutreffen sind. Im Folgenden werden diese erläutert.

4.2.1 JMeter

Um die definierten Testszenarien durchzuführen, wurde das Open Source Tool Apache JMeter (v.3.3) genutzt. Es gibt eine ganz Reihe von Lasttest- und Performancetest-Software auf dem Markt wie HP LoadRunner, Silk Performer, LoadUI und viele mehr. JMeter wurde ausgewählt, da es ein kostenloses und das am meisten genutzte Last- und Performance-Tool ist. Das Tool unterstützt eine Reihe verschiedener Protokolle, unter anderem REST, was für dieses Experiment genutzt wurde. Die Testszenarien wurden im "Non-GUI oder "headlessMode durchgeführt. Dieser dient dazu, intensive Testreihen in einer CLI-Umgebung

durchzuführen. Um die Tests durchzuführen, wurde für das Speichern der Ergebnisse folgendes Skript erstellt:

```
1 @echo off
2 for /f "tokens=2 delims==" %%a in ('wmic OS Get localdatetime /value') do set
3     "dt=%%a"
4 set "YY=%dt:~2,2%" & set "YYYY=%dt:~0,4%" & set "MM=%dt:~4,2%" & set "DD=%dt:~6,2%"
5 set "HH=%dt:~8,2%" & set "Min=%dt:~10,2%" & set "Sec=%dt:~12,2%"
6
7 set "datestamp=%YYYY%-%MM%-%DD%_%HH%-%Min%-%Sec%"
8 set "fullstamp=%YYYY%-%MM%-%DD%_%HH%-%Min%-%Sec%"
9
10 jmeter -n -t C:/sandbox/dinf/dinf/loadtest/jmeter/LoadTest-TIMED.jmx -l
    C:/sandbox/dinf/dinf/loadtest/jmeter/testlog-%fullstamp%.jtl
```

JMeter



JMeter ist ein Open Source Tool und kann direkt bei Apache heruntergeladen werden.

<http://jmeter.apache.org/>

4.2.2 Test Szenarien

Es wurden verschiedene Szenarien definiert, die einerseits einen möglichen Anwendungsbereich von OSRE abbilden könnten, andererseits sich an den speziellen Implementationen oder Limitierungen der OSRE orientieren. Als Beispiel verfügt Apache PDF Box nicht über eine integrierte Schnittstelle, um Tabellen zu implementieren. Im Folgenden werden diese Szenarien detailliert vorgestellt.

PDF Resultate



Die originalen PDFs können auf GitHub oder im Anhang nachgeschlagen werden: <https://github.com/denisbittante/dinf/tree/master/doc/results/pdf>

Szenario 1 - Aktivität

Im ersten Szenario (siehe Abbildung 4.2) ging es darum, einen Report wiederzugeben, der in einem Referenzprojekt genutzt wird. Die Anforderung dabei war, einen einfachen Report zu erstellen, der eine Reihe von Aktivitäten oder Terminen abbildet. Dabei wurde darauf geachtet, dass alle diese Anforderung erfüllen können. Einerseits wurden die Schriftart

4 Evaluation

<p>Aktivitäts Titel</p> <p>12.02.2017 16:00 - 12.02.2017 16:00</p> <p>Die Sommerabende waren lang. Noch war's nicht dunkel geworden. Toms Pfoten verkrummt plötzlich. Ein Fremder stand vor ihm, ein Junge, nur vielleicht einen Zoll größer als er selbst.</p> <p>Die Erinnerung eines Fremden irgendwoher Alters oder Geschlechtes war ein Ereignis in dem armen, kleinen Städtchen St. Petersburg. Und dieser Junge war noch dazu keiner gekleidet, – sauber gekleidet an einem Wochenende! Das war einfach unglaublich übertrieben! Seine Mütze war ein netter, kleiner Druck. Seine Hose war eine dicke, dreckige Tuchjacke mit und tatsächlich ohne Hosen ohne Flecken. Schuhe hatte er an, Schuhe, und es war doch keine einzige Stelle, an der sie sich am Boden abdrückte. Und seine Nase rumpfte über das weiße Tuch geschürrten. Er hatte so etwas Zivilisiertes, so etwas Städtisches an sich, das Tom in die innere Seele schnitt. Ja mehr er dieses Wunder von Eleganz anstandt, je mehr er die Nase rumpfte über den verblümten Schneidezahn, und rumpfte und rumpfte und seine Augen funkelten. Keiner der Jungen sprach. Wenn der sich bewegte, bewegte sich auch der andere, aber immer nur gebückt im Kreise herum. Be standen sie einander gegenüber, Angesicht zu Angesicht, Auge in Auge.</p> <p>Ort Musterstrasse 12 9000 St. Gallen Schweiz</p> <p>Verantwortlicher Denis Bitante</p> <p>Helfer Lars Hauser, Someoneelse</p> <p><small>Erstellt von: Denis Bitante</small></p> <p><small>Seite 1/2</small></p> <p><small>Seit.Feu.10.01.5843.CET.2018</small></p>	<p>Aktivitäts Titel</p> <p>12.02.2017 16:00 - 12.02.2017 16:00</p> <p>Die Sommerabende waren lang. Noch war's nicht dunkel geworden. Toms Pfoten verkrummt plötzlich. Ein Fremder stand vor ihm, ein Junge, nur vielleicht einen Zoll größer als er selbst.</p> <p>Die Erinnerung eines Fremden irgendwoher Alters oder Geschlechtes war ein Ereignis in dem armen, kleinen Städtchen St. Petersburg. Und dieser Junge war noch dazu sauber gekleidet, – sauber gekleidet an einem Wochenende! Das war einfach unglaublich übertrieben! Seine Mütze war ein netter, kleiner Druck. Seine Hose war eine dicke, dreckige Tuchjacke mit und tatsächlich ohne Hosen ohne Flecken. Schuhe hatte er an, Schuhe, und es war doch keine einzige Stelle, an der sie sich am Boden abdrückte. Und seine Nase rumpfte über den verblümten Schneidezahn, und rumpfte und rumpfte und seine Augen funkelten. Keiner der Jungen sprach. Wenn der eins sich bewegte, bewegte sich auch der andere, aber immer nur gebückt im Kreise herum. Be standen sie einander gegenüber, Angesicht zu Angesicht, Auge in Auge.</p> <p>Ort Musterstrasse 12 9000 St. Gallen Schweiz</p> <p>Verantwortlicher Denis Bitante</p> <p>Helfer Lars Hauser, Someoneelse</p> <p><small>Erstellt von: Denis Bitante</small></p> <p><small>Seite 1/2</small></p> <p><small>Seit.Feu.10.01.5843.CET.2018</small></p>	<p>Aktivitäts Titel</p> <p>12.02.2017 16:00 - 12.02.2017 16:00</p> <p>Die Sommerabende waren lang. Noch war's nicht dunkel geworden. Toms Pfoten verkrummt plötzlich. Ein Fremder stand vor ihm, ein Junge, nur vielleicht einen Zoll größer als er selbst.</p> <p>Die Erinnerung eines Fremden irgendwoher Alters oder Geschlechtes war ein Ereignis in dem armen, kleinen Städtchen St. Petersburg. Und dieser Junge war noch dazu sauber gekleidet, – sauber gekleidet an einem Wochenende! Das war einfach unglaublich übertrieben! Seine Mütze war ein netter, kleiner Druck. Seine Hose war eine dicke, dreckige Tuchjacke mit und tatsächlich ohne Hosen ohne Flecken. Schuhe hatte er an, Schuhe, und es war doch keine einzige Stelle, an der sie sich am Boden abdrückte. Und seine Nase rumpfte über den verblümten Schneidezahn, und rumpfte und rumpfte und seine Augen funkelten. Keiner der Jungen sprach. Wenn der eins sich bewegte, bewegte sich auch der andere, aber immer nur gebückt im Kreise herum. Be standen sie einander gegenüber, Angesicht zu Angesicht, Auge in Auge.</p> <p>Ort Musterstrasse 12 9000 St. Gallen Schweiz</p> <p>Verantwortlicher Denis Bitante</p> <p>Helfer Lars Hauser, Someoneelse</p> <p><small>Erstellt von: Denis Bitante</small></p> <p><small>Seite 1/2</small></p> <p><small>Seit.Feu.10.01.5843.CET.2018</small></p>
--	---	---

Abbildung 4.2: PDF-Resultate Szenario 1 (v.l.n.r iText, JasperReports, Apache PDFBox)

gewählt und andererseits das Format der Überschriften sowie die Reportgrösse (DIN A4) definiert. Alle Implementationen mussten auch die PDF-Metadaten befüllen, wie die Felder Stichwörter, Autor und Titel. Eine weitere Anforderung für das erste Szenario waren Fussnoten, die es zu implementieren galt.

Die Informationen, die auf den Reports dargestellt werden, spiegeln die Requests wieder. Jeder Eintrag im Request löst dabei eine neue Seite im Report aus, da Apache PDFBox nicht in der Lage war, Floating-Layouts zu erstellen. Somit werden 10 bis 30 Seiten pro Test generiert. Dieser Report soll frei von rechenintensiven Kalkulationen sein, damit gezeigt werden kann, was die s unter optimalen Bedingungen leisten könnten.

Szenario 2- Zeitplan

Im zweiten Szenario wurde ein eher aufwendiges Design erstellt. Der Report bildet einen Zeitplan ab mit verschiedenen Aktivitäten oder Terminen, gruppiert nach Tag. Jeder Tag besitzt dabei ein Datum und einen passenden Titel. Diese Titelzeile galt es farblich hervorzuheben (siehe Abbildung 4.3). Was das Layout anbelangt, wurden verschiedene Anforderungen definiert, um es etwas anspruchsvoller zu gestalten. Die Tagesabschnitte besitzen je eine Tabelle mit den einzelnen Aktivitäten, auf drei Spalten aufgeteilt. Wird im Request das Flag `isSubtitle` als 'true' gesendet, wird der Untertitel ebenfalls farblich im gelb-orange Farbton hervorgehoben. Auch Kursiv und Fett im Schriftzug wurde eingesetzt. Das PDF wurde auch hier mit den nötigen Metadaten versehen. An dieser Stelle ist hervorzuheben, dass als Ziel nicht eine perfekte Darstellung der PDF-Reporte im Vordergrund

4.2 Test Durchführung

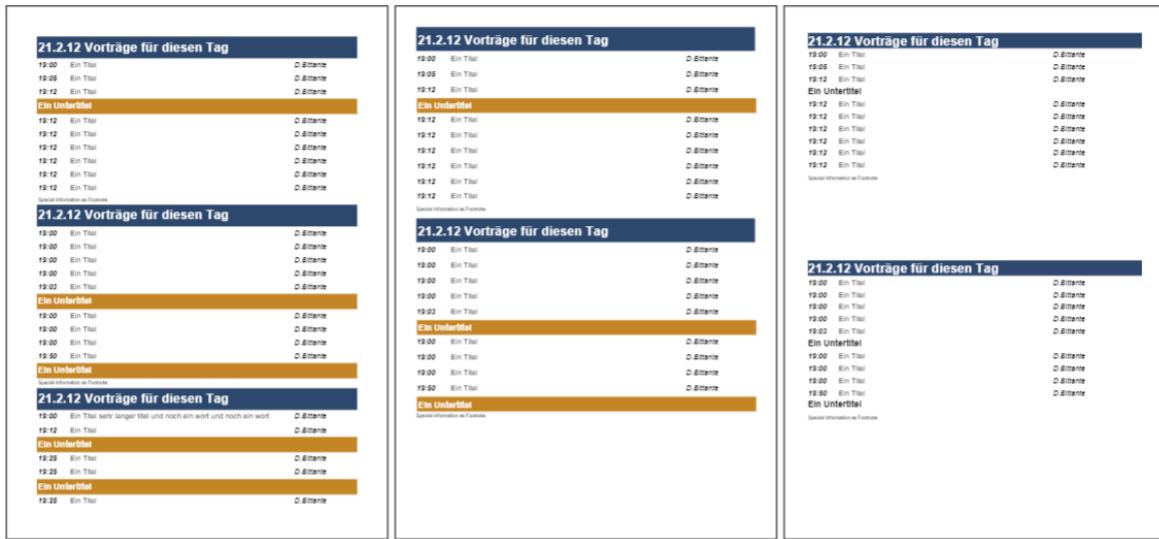


Abbildung 4.3: PDF-Resultate Szenario 2 (v.l.n.r iText, JasperReports, Apache PDFBox)

stand, sondern ein möglichst ähnliches und vergleichbares Ergebnis zu erzielen. Auch hier sind Unterschiede festzustellen. Als grober Unterschied ist ApachePDFBox aufzuführen, bei der die Hervorhebung des Untertitels in Farbe zu aufwendig wurde. Da der Hintergrund der Titelzeile als Box gezeichnet wird, mussten die entsprechenden Koordinaten angegeben werden. Um den Hintergrund eines Untertitels anzugeben, hätte man die Berechnung der Koordinaten in Bezug auf die Font-Grösse durchführen müssen, was nicht mehr zielführend war.

In diesem Anwendungsfall gilt es zu prüfen, wie eine OSRE mit komplexem Seitenaufbau zurechtkommt und wie sich das auf die Performance auswirkt und vor allem, wie die Ressourcen dabei genutzt werden. Dabei soll der Seitenaufbau komplizierter sein. Die Datenmengen werden im Test zwischen 10 und 30 Datensätzen sein, was zu einer PDF-Länge von vier (iText) bis maximal 15 Seiten führt.

Szenario 3 - Kontaktliste

In diesem Anwendungsfall soll gezeigt werden, welche OSRE am besten geeignet wäre, Bilder in einem Report zu verarbeiten. Dieser Anwendungsfall generiert zwar nur Reports, die ein bis zwei Seiten lang sind, diese müssen aber dennoch möglichst schnell generiert werden. Der definierte Anwendungsfall kann in der Abbildung 4.4 gesehen werden. Das definierte Ziel war eine Übersicht der Kontakte, die z.B. im Zusammenhang mit den Aktivitäten eingeladen hat. Es wurde definiert, dass diese Übersicht erneut die nötigen Metadaten befüllt, auch Fusszeilen auf den PDFs dürfen nicht fehlen. Das interessante in diesem Anwendungsfall

4 Evaluation

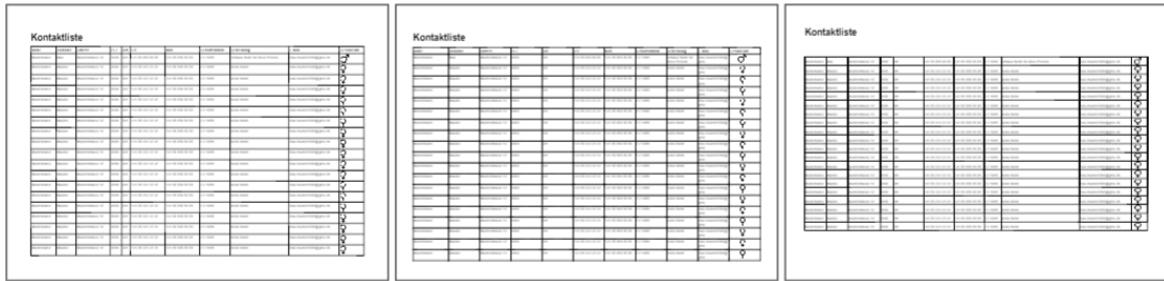


Abbildung 4.4: PDF-Resultate Szenario 3 (v.l.n.r iText, JasperReports, Apache PDFBox)

war die Verarbeitung der Bilder. Die Bilder waren in einer hohen Auflösung abgelegt und mussten skaliert und gegebenenfalls komprimiert werden. Des Weiteren wurde definiert, dass sich die Seite im Querformat zu orientieren hat und der Titel nur auf der ersten Seite anzuzeigen ist.

Die zu erzeugenden Tabellen wurden von den s verschieden implementiert, wie in der Einführung bereits angesprochen wurde. In iText lässt die Tabellenfunktion einzelne Zellen erstellen und mit JasperReports konnte die Tabelle als Subreport oder sogar als Datensatz formatiert werden. Apache PDFBox war in dieser Hinsicht etwas primitiv. Die Tabelle wurde in der Grösse berechnet und jedes Feld wurde einzeln auf der Arbeitsfläche hinzugefügt. Anschliessend wurden die Felder mit gezeichneten Strichen umrandet.

4.3 Konsolidierung

In diesem Schritt der Evaluation geht es darum, die Daten zu vereinheitlichen und zu konsolidieren.

4.3.1 Log Datei

Die Daten, die Heroku niederschreibt, sind unter anderem die eingehenden Requests, der Dyno Load und der Memoryverbrauch. Diese Daten werden als Zeichenketten in die Loggs geschrieben. Nicht gesteuert von den Aktivitäten auf dem Dyno, sondern aufgrund eines Timers werden diese Daten aus einer Probe des Dynos gelesen und in die Logs persistiert. Um diese Daten auswerten zu können, wurde ein eigener Log Parser geschrieben, mit welchem es möglich ist, diese Logs zu finden und in einer eigenen kommasseparierten Datei zu speichern.



Log Extractor Source Code

Der Log Extractor kann hier gefunden werden: <https://github.com/denisbittante/dinf/tree/master/loadtest/utils/src/ch/ffhs/dinf/utils/log>

4.3.2 JMeter Testresultate

Die Ergebnisse von JMeter können wiederum im selben Tool im Gui-Modus gelesen und ausgewertet werden. Dazu bietet JMeter verschiedene Auswertungsmethoden an. Einerseits können Plug-In's weitere Aufbereitungsmethoden zur Verfügung stellen, oder sonst sind mit JMeter bereits ein Dutzend Aggregatoren und grafische Tools mit dabei. Des Weiteren wurde Microsoft Excel genutzt, um die Daten in grafischer Form aufzubereiten.



Konsolidierte Daten

Die Konsolidierten Daten können hier heruntergeladen werden: <https://github.com/denisbittante/dinf/tree/master/loadtest/results>

5 Resultate

5.1 Performance

Die Tests ergaben eine aufschlussreiche Datenmenge in Hinblick auf die Performance. Dabei sind nicht allein der Durchsatz und die Latenzzeit wichtige Indikatoren. Es gilt ebenfalls zu analysieren, wie die Ressourcen auf der PaaS genutzt wurden. Dabei werden die Daten miteinander verglichen und mögliche Ursachen für die Performance aufgezeigt.

5.1.1 Durchsatz nach Anfragen

Wie in der Abbildung 5.1 ersichtlich wird, haben alle OSRE verschiedene Durchsatzraten erzeugt.

Szenario 1 und 2

Der höchste Durchsatz gelang mit Apache PDFBox. Der maximale Durchsatz in der Testreihe betrug 151 Anfragen pro Sekunde. Dies wurde beim Szenario 1 mit der Rate von 50 virtuellen Usern erreicht. Apache PDFBox hat bei Durchsatz und Verarbeitungsgeschwindigkeit in den ersten beiden Szenarien die höchste Rate im Vergleich zu iText und JasperReports erreicht. Der Durchsatz lag beim ersten und zweiten Szenario mit Apache PDFBox bei über 100 Anfragen pro Sekunde, ausser bei den Szenarien 1b und 2b, bei denen die zu verarbeitenden Daten verdreifacht wurden. iText zeigt ähnliche Ergebnisse, dies mal mit dem Maximal Durchsatz von 60 Anfragen pro Sekunde im Szenario 2 mit ebenfalls 50 virtuellen Usern. Der Durchsatz von iText und Apache PDFBox schwankt stark. Ein Vergleich vom besten Ergebnis (Szenario 1c) zum schwächsten (Szenario 1b) mit Apache PDFBox zeigt eine Verschlechterung des Durchsatzes von bis zu 60.9%. Auch bei iText ist eine Verminderung des Durchsatzes festzustellen, die bis zu 83% betrug. Hingegen blieb der Durchsatz bei JasperReports vergleichsweise konstant. JasperReports zeigt bei allen Szenarien einen Durchsatz zwischen 11 und und 17 Anfragen pro Sekunde und eine robustere, jedoch langsame Implementierung des Service.

Szenario 3

Das dritte Szenario zeigt eine rapide Abnahme der Durchsätze. Alle drei OSREs bringen den Durchsatz nicht über 15 Anfragen pro Sekunde. Das bedeutet, dass im besten Fall das Szenario 3 von JasperReports 52'200 mal in einer Stunde verarbeitet werden kann. iText würde etwa 34'200 PDF in der gleichen Zeit generieren und Apache PDFBox gerade mal 18'360 PDFs liefern. Apache PDFBox hat in diesem Szenario die langsamste Verarbeitung mit 1.8 Anfragen pro Sekunde, knapp gefolgt von iText mit 3.1 Anfragen pro Sekunde.

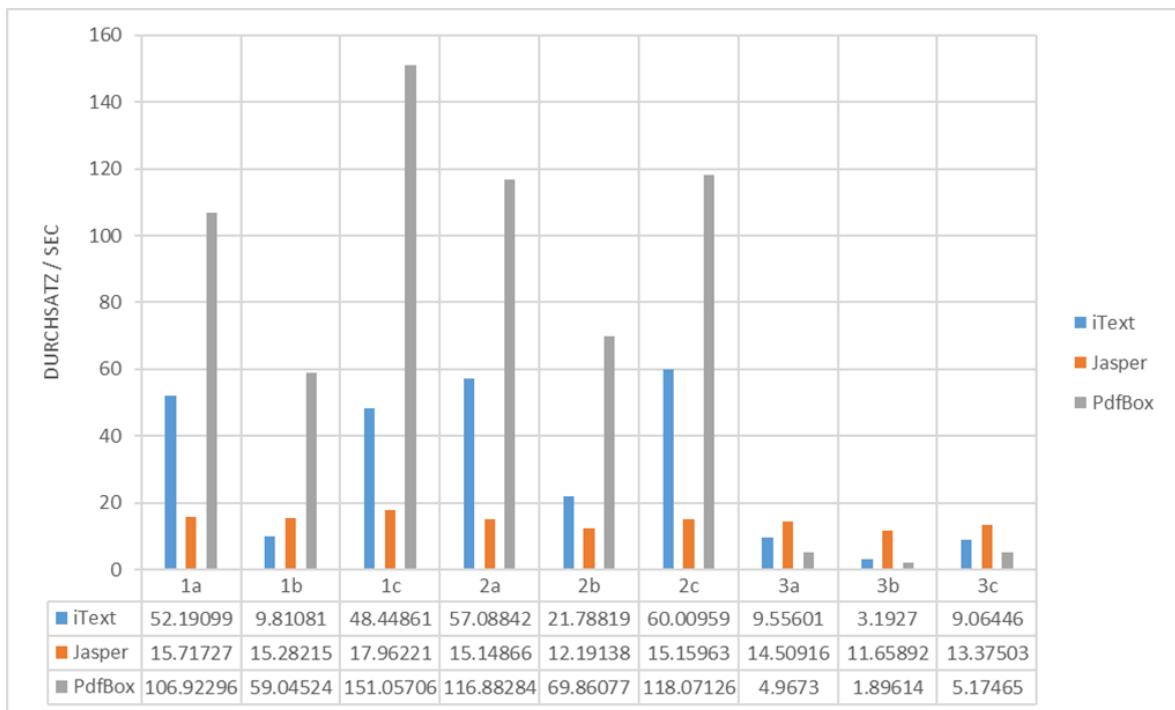


Abbildung 5.1: Vergleich - Durchsatz nach Szenario

5.1.2 Durchsatz nach Bytes

Die Abbildung 5.2 zeigt, dass der Durchsatz in Bytes über die Testdauer hinweg nicht einbricht. Der Durchsatz nach Bytes scheint nur bei JasperReports tief zu bleiben. Der Durchsatz nach Bytes scheint auch bei Apache PDFBox und iText beim Szenario 3 zu steigen, wobei der Durchsatz der Antworten nachlässt. Analysiert man die Grösse der Antworten, wird auch klar warum.

JasperReports generiert im Vergleich zu den anderen OSREs weitaus kleinere PDFs. Die Kontaktliste, eine Liste mit Kontaktdaten und einem Gender-Symbol, die im Szenario 3

5 Resultate

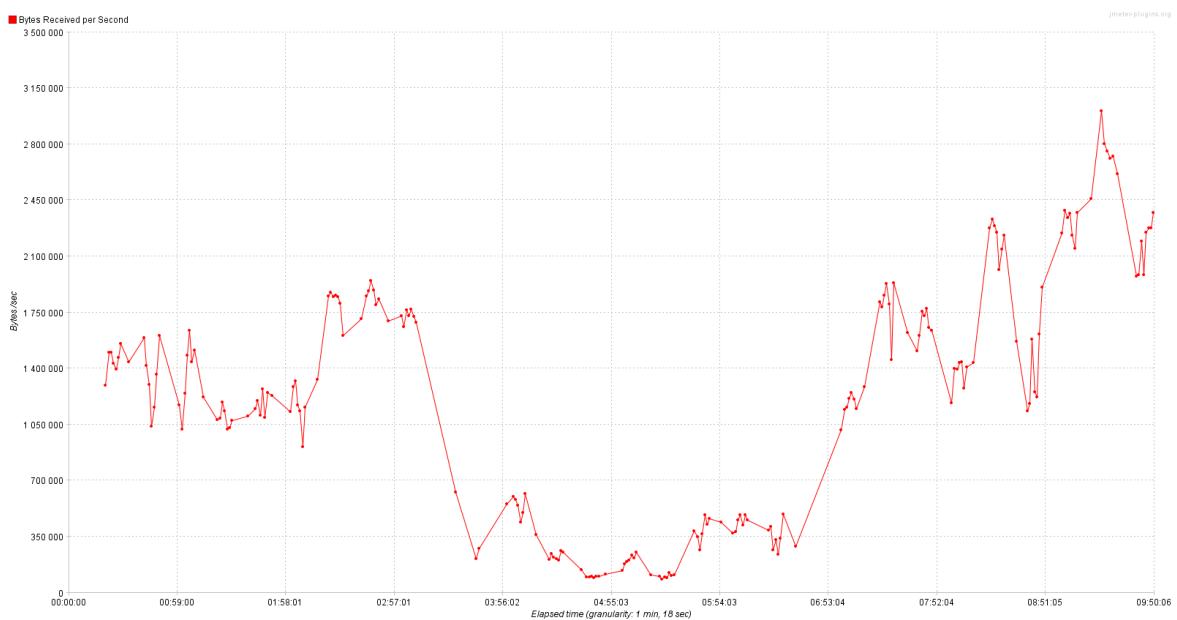


Abbildung 5.2: Durchsatz in Bytes

generiert wird, verarbeitet JasperReports besser als die anderen OSREs. Es scheint, als ob die Bilder referenziert und komprimiert würden. Dies ist mithilfe der vorkompilierten Templates möglich. Das mit JasperReports generierte PDF ist etwa 46KB gross, das mit Apache PDF-Box generierte PDF ist etwa 730KB gross. Etwas besser ist die Komprimierung bei iText, mit dem das PDF etwa 350 KB gross wird. Dies bedeutet zwar, dass der Durchsatz nicht vergrössert wurde, doch die generierten Files nutzten dennoch die Bandbreite des verfügbaren Netzwerkes. Wie gross die Bandbreite eines Webservice ist und welche Netzwerkverbindung den Engpass darstellte, konnte nicht festgestellt werden.

5.1.3 Latency

Die Latenzzeit, die in der Abbildung 5.3 dargestellt wird, ist die Zeit, die der Service gebraucht hat, um die Anfrage zu bearbeiten, samt Netzwerkzeit.

Diese Grafik korreliert klar mit der Analyse der Durchsätze. Der Durchsatz steigt umgekehrt proportional zur gemessenen Latenzzeit.

Auch hier kommen iText und Apache PDFBox bei den meisten Szenarien mit weniger als einer Sekunde aus. Doch es sind auch Ausnahmen festzustellen, wie beim Szenario 1b bei iText und im Szenario 3 bei Apache PDFBox. Diese Latenzzeiten sind keine Aussage zur Netzwerklatenz, die reine Zeit, die verbraucht wird, bis eine Nachricht vom Sender zum Empfänger übermittelt wurde. Diese Zeit gibt die reine Antwortzeit des Service wieder.

5.1 Performance

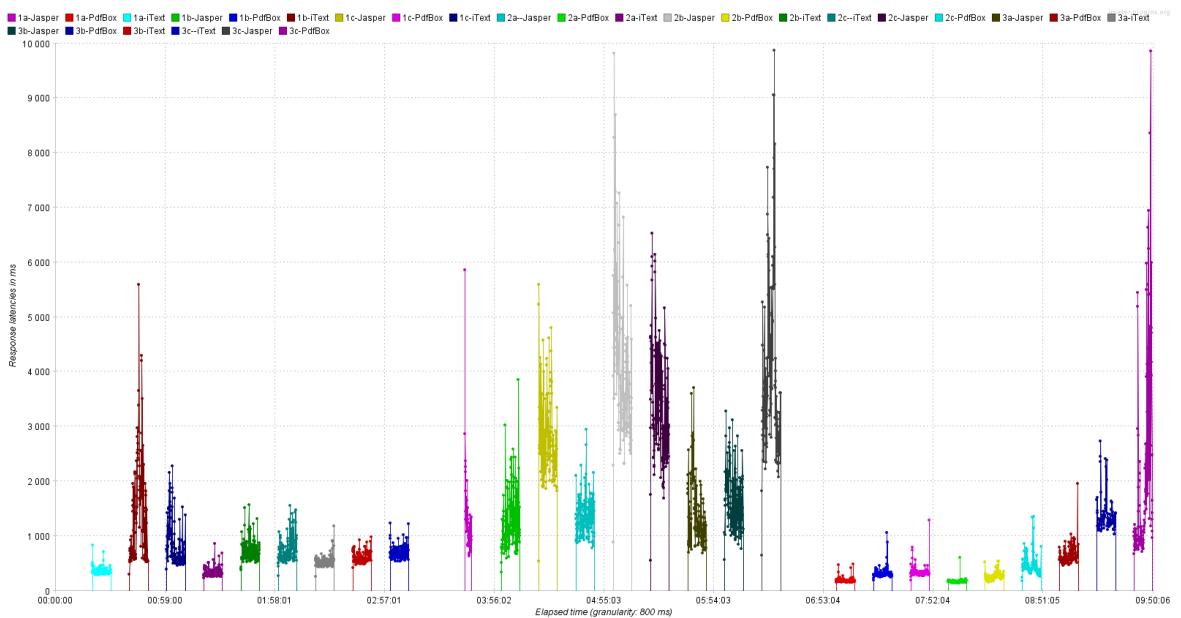


Abbildung 5.3: Latenzzeit im Testzyklus

Auch bei Abbildung 5.4 vergrößert sich die Latenz wegen der Verarbeitungszeit während des Szenarios 3. JasperReports zeigt dabei eine überdurchschnittliche Latenzzeit während der ersten beiden Szenarien.

5.1.4 Verfügbarkeit

Der Service soll auch über die Zeit hinweg möglichst verfügbar bleiben. Dies ist ebenfalls ein wichtiger Performance-Indikator. Während der Tests hat keiner der Services über den Testlauf von rund drei Stunden einen Absturz gehabt oder ist unerreichbar gewesen. Die Prototypen waren im Verlauf der Tests meist aktiv geblieben, was aber nicht heisst, dass ein User diese auch als verfügbar empfunden hätte. Hätte man diese Prototypen in einer Produktionsumgebung eingesetzt, wäre der User wohl des Öfteren mit längeren Wartezeiten konfrontiert gewesen. Die Latenzen, die im vorhergehenden Kapitel erwähnt wurden, wären auch hier ein Thema. In der Abbildung 5.5 wird ersichtlich, wie viel Prozent der Anfragen an die entsprechenden Services länger gedauert hätten als 2 Sekunden.

Nach Miller [7, Seite 270] sind Verzögerungen erst dann ein Problem, wenn diese mit dem Fluss der spezifischen Arbeit zusammenkommen. Ein Betriebssystem muss immer in weniger als 0.1 Sekunden antworten, damit es den Arbeitsfluss nicht behindert. Auch hat ein Schreibprogramm in weniger als einer Sekunde zu reagieren, auch wenn Graphiken verarbeitet werden, damit der User in seinem kreativen Prozess nicht behindert wird. Weiter sind die

5 Resultate

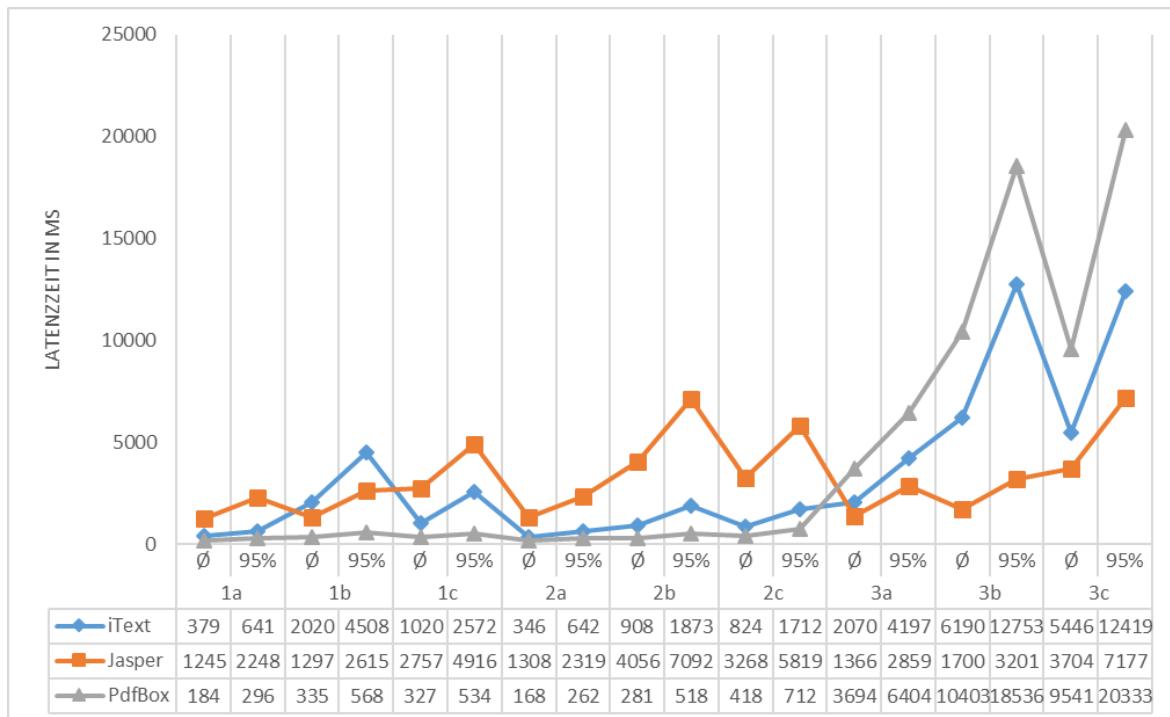


Abbildung 5.4: Latenzzeit nach Szenario und OSRE

Zeitspannen von 2 bis 4 Sekunden ebenfalls eine wichtige Grösse. Bei komplexen Arbeiten, bei denen der User sich Informationen merken muss, ist es wichtig, diese Antwortzeit so gering wie möglich zu halten.

In der Abbildung 5.5 wird gezeigt, dass sich besonders JasperReports für eine Online-Applikation nicht wirklich eignete. Vier von neun Szenarien benötigen meist mehr als 2 Sekunden, um eine initiale Antwort zu senden. Die Latenzzeit beträgt bei mehr als 85% der Antworten mehr als 2 Sekunden. Dies wäre für den Online-Benutzer zu lange, wenn diese Daten für seinen Arbeitsprozess nötig wären.

JasperReports liefert 99% der Antworten innerhalb von 10 Sekunden, was für eine Batchverarbeitung akzeptabel wäre. Dennoch haben auch iText und Apache PDFBox bei einer SLA von zwei Sekunden in einigen Fällen Schwierigkeiten. iText hat bei der Verarbeitung des Szenarios 1b eine Verfügbarkeit von 77% erreicht, was zu tief ist für eine Online-Verarbeitung. Bei den übrigen Szenarien, mit der kleinen Ausnahme des Szenarios 1c, das bei 96% angelangt ist, konnte bei 99% eine Antwortzeit von unter zwei Sekunden erreicht werden.

Apache PDFBox zeigt hingegen wieder, dass die Verarbeitung der Reports im Szenario 3b und 3c oft eine Geschwindigkeit erreicht, die der User nicht mehr als angenehm oder akzeptabel empfinden würde.

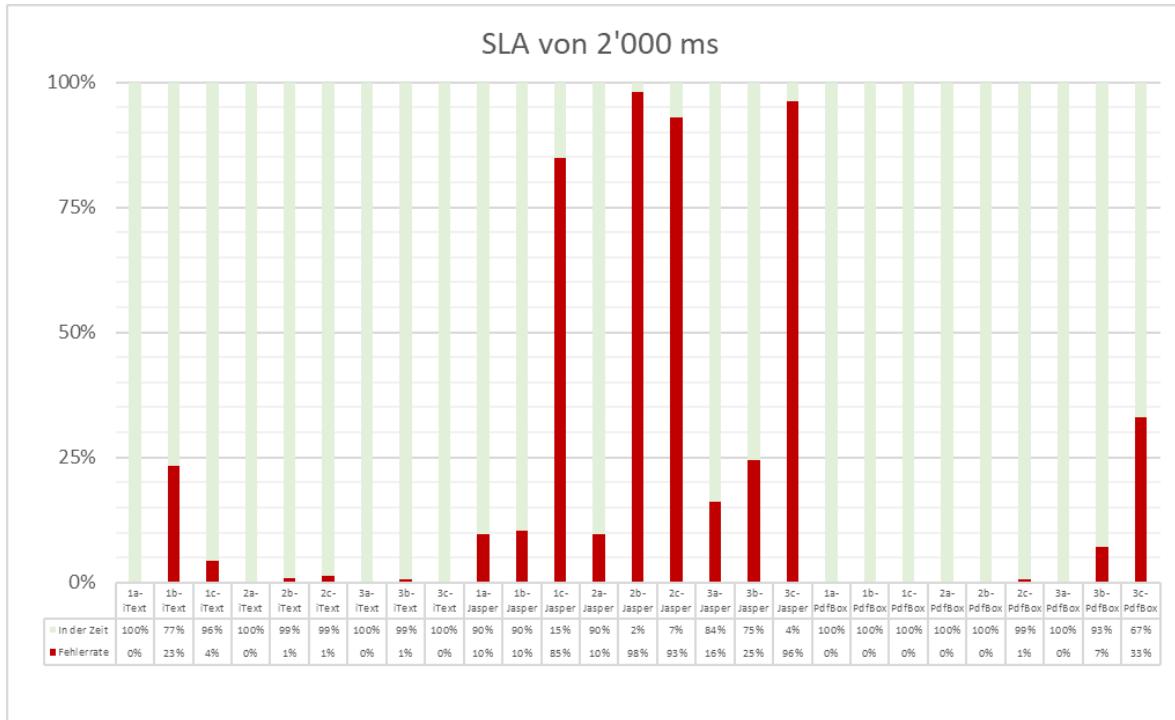


Abbildung 5.5: Fehlerrate nach Latenzzeit über 2000ms

5.2 Resourcen

Welche Auswirkung die Test auf dem Webservern haben wird in den folgenden Kapitel behandelt.

5.2.1 Arbeitsspeicher

Aus den Performance-Messungen konnte der Verbrauch auf der PaaS geloggt werden und ist in der Abbildung 5.6 zu sehen. Die Memoryauslastung steigt bei der Ausführung der Performancetests und sinkt bei der Ausführung des GC oder einem Neustart des Webservers (Dynos). Die Dynos sind als Standard-X2 definiert, was bedeutet, dass die maximale Auslastung auf den Memory RSS auf 1024MB beschränkt ist. Dies bedeutet, dass die Auslastung auf dem Memory begrenzt ist und jede weitere Verarbeitung auf den SWAP ausgelagert wird. Die Abbildung 5.6 zeigt diese Phänomene. iText nutzte nur wenig SWAP und konnte in den Performancetests mit RSS Memory meist alle Anfragen bearbeiten. Ein Teil der Anfragen wurde dennoch im Szenario 3 im SWAP ausgelagert. Nach einem Restart des Webservices wurde JasperReports genutzt. Diese ORSE nutzte bereits das verfügbare Memory im ersten Szenario. Das Memory auf dem SWAP konnte über alle Szenarien hinweg nicht freigegeben

5 Resultate

werden, was zu einem Aufblasen des Memoryverbrauchs führte. JasperReports erreichte im Testlauf einen totalen Memoryverbrauch von 1389.32MB. Apache PDFBox erreichte im Testlauf die Memory-Quota kaum.

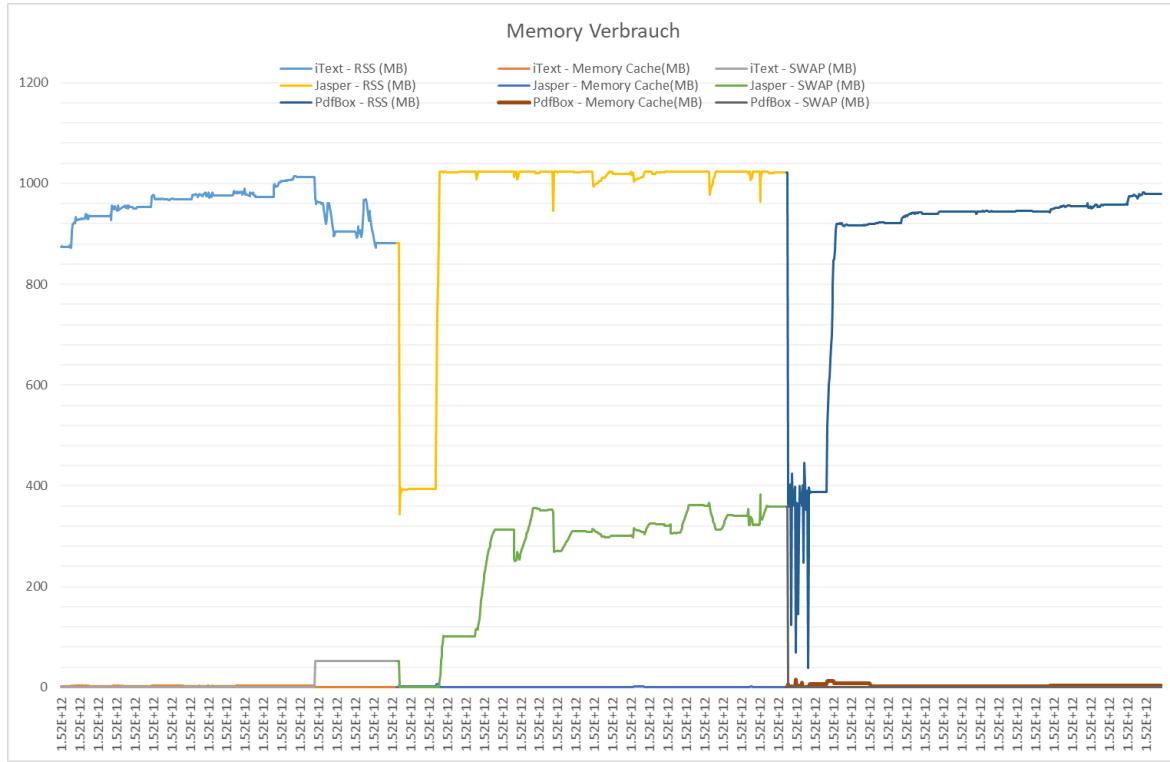


Abbildung 5.6: Memoryverbrauch Testlauf

5.2.2 Prozessor Last

Der Verbrauch des CPU ist eine etwas schwierige Messgrössse. Die Platform as a Service hat diese Daten mit der Stichprobengrösse von einer Minute in die Logs geschrieben.

Dabei ist natürlich zu erkennen, dass sich im Textzyklus von iText die Auslastung je nach Szenario sehr stark unterscheidet. Z.B. sind während des zweiten Szenarios durchschnittlich zwischen vier und sieben Prozessoren im Status 'Ready' pendent gewesen. Hingegen sind im dritten Szenario fast immer alle Prozessoren in Ausführung gewesen, das bedeutet eine fast optimale Auslastung des CPUs mit wenigen wartenden Prozessen. Diese sind wohl so entstanden, da die Prozesse im dritten Szenario langläufiger waren und somit wenig neue Anfragen verarbeitet wurden.

JasperReports hat den CPU eher gefordert und hat über den ganzen Zeitraum viele Prozesse gebraucht. Wie intensiv die Verarbeitung von JasperReports im Vergleich zu den

5.3 Reporting Engines - Verhalten

anderen OSREs ist, wird hier klar hervorgehoben

Apache PDFBox ist hier als sehr sparsam zu erkennen. Es wurden maximal fünf Prozesse ausgelagert.

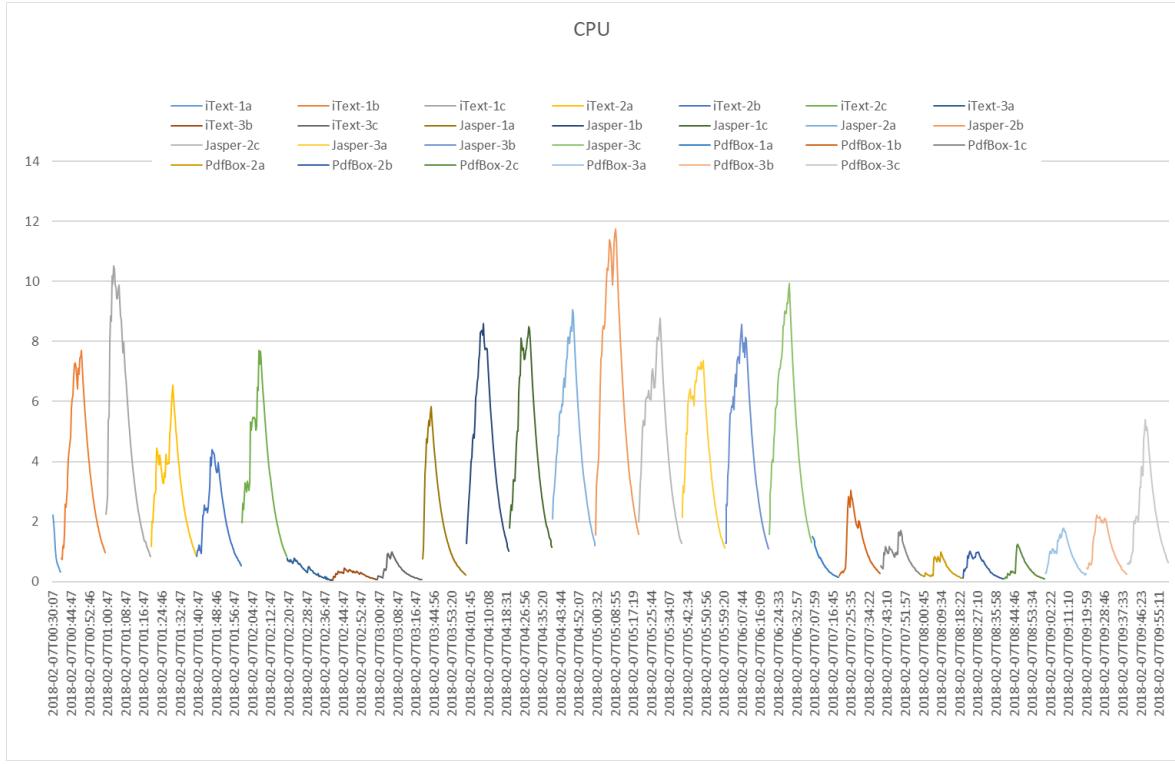


Abbildung 5.7: Vergleich - Durchsatz nach Szenario

5.3 Reporting Engines - Verhalten

Die OSREs haben sich während des Tests in Bezug auf die Lastveränderung verschieden verhalten. In diesem Kapitel sollen die Veränderungen von Requestgrößen und die Zunahme von virtuellen Usern (VU) weiter aufgezeigt werden.

5.3.1 Requestgrösse

Im Vergleich zu den Szenarien 1a, 2a und 3a wurden bei den Szenarien 1b, 2b und 3b die Datensätze für die Verarbeitung verdreifacht. Wie den Abbildungen 5.8 entnommen werden kann, hat dies dazu geführt, dass sich alle Antwortzeiten und Durchsätze verschlechtert haben

5 Resultate

OSRE, iText und Apache PDFBox haben sich in der gleichen Art und Weise verschlechtert. iText verzeichnet hier Durchsatzeinbrüche. Die ersten zwei Szenarien fallen je von zwischen 50 und 60 auf fast 10 und 21 Anfragen pro Sekunde.

Hier sei noch einmal hervorgehoben, dass diese Daten aufgrund eines Testlaufs ermittelt wurden. Es kann in jedem Testlauf das gleiche Phänomen beobachtet werden, dennoch sind diese Daten nicht stellvertretend oder absolut zu betrachten. Größere Ausreisser wurden in anderen Testläufen aufgezeichnet.

5.3.2 Virtuelle User

Die Prototypen wurden von den initialen 20 VU im Szenario 'a' weiter gefordert, als 30 weitere VUs hinzugefügt wurden, um eine erhöhte Nutzerinteraktion zu simulieren. Wie die s dabei reagiert haben, kann in der Abbildung 5.9 gesehen werden. In den Abbildungen werden die Veränderungen des Durchsatzes mit dem 95 Perzentil der Antwortzeit verglichen. Dabei ist nur massgebend, wie sich die s im Durchschnitt verhalten.

Einige der interessanten Erkenntnisse in diesem Zusammenhang können bei Apache PDFBox entdeckt werden. Obwohl die Nutzerlast steigt, steigt auch der Durchsatz im Szenario

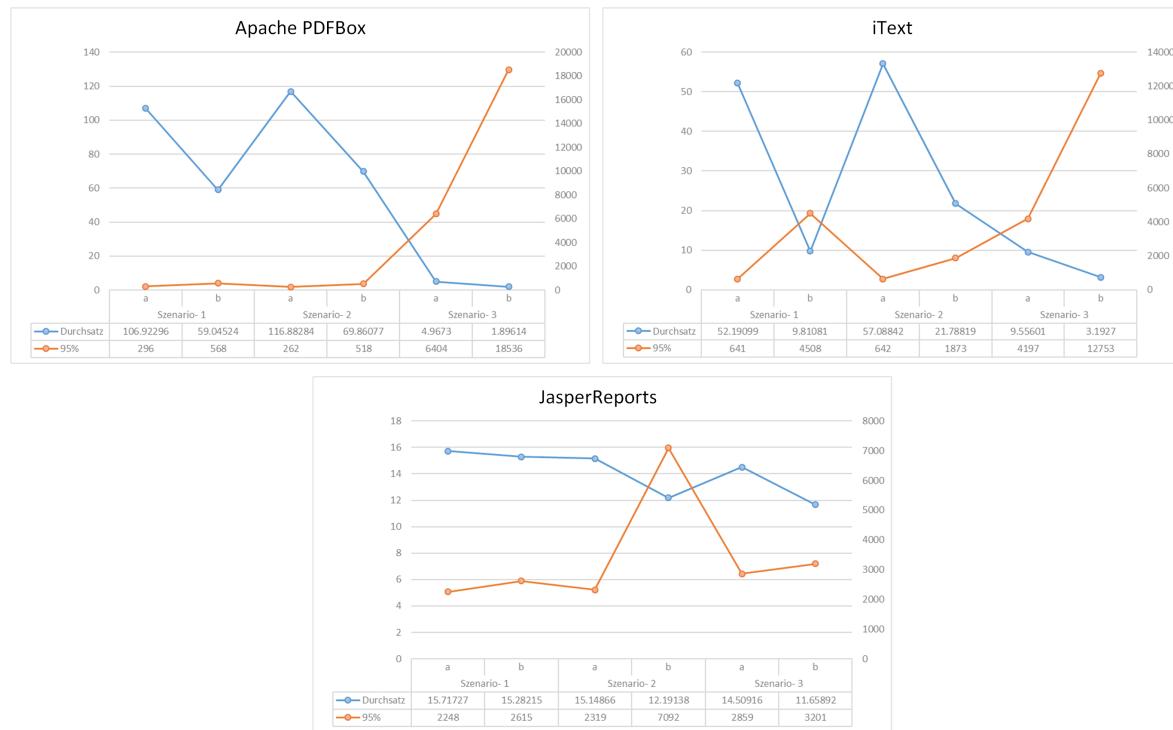


Abbildung 5.8: Vergleich Baseline mit grösserer Datenmengen

5.3 Reporting Engines - Verhalten



Abbildung 5.9: Vergleich Baseline mit mehr virtuellen Usern

1 von 106 auf 151 Anfragen pro Sekunde. Die Antwortzeit verdoppelt sich aber dabei. Ähnliches ist auch im zweiten Szenario zu beobachten. Beim Kreuzvergleich mit der Zunahme der zu verarbeitenden Daten (Abbildung 5.8) sieht man klar, dass diese im Durchsatz einen deutlichen Einbruch erleben, aber die Antwortzeit gleich bleibt. Apache PDFBox hat klar Mühe mit dem Szenario 3. Die Antwortzeit verdreifacht sich dabei, wobei der Durchsatz gleich bleibt.

iText scheint in der Implementation der ersten zwei Szenarien ein ähnliches Verhalten unter Last zu erzeugen. Die Antwortzeit wird im ersten Szenario viermal langsamer, im Szenario 2 dreimal langsamer. Auch iText schafft es, seinen Durchsatz zu verbessern, und zwar werden drei Antworten mehr pro Sekunde verarbeitet. Szenario 3 hat ähnliche Probleme wie Apache PDFBox. Die Antwortzeiten verdreifachen sich und der Durchsatz bleibt auch bei iText etwa konstant.

JasperReports verschlechtert sich in allen Antwortzeiten, aber nicht ganz so extrem wie iText und Apache PDFBox. In allen Szenarien verschlechtern sich die Antwortzeiten. Bei allen Szenarien wurde JasperReports etwa doppelt so langsam. Was den Durchsatz anbelangt, blieb dieser fast konstant, im Szenario 1 wurde dieser ebenfalls etwas höher (2 Anfragen pro Sekunde mehr).

5.4 Ursachenanalyse

Im Szenario 3 konnte man den Rückgang von iText und Apache PDFBox erkennen. Es lässt sich anhand der Ergebnisse der verschiedenen Metriken die Schlussfolgerung ziehen, dass der Durchsatz nicht durch eine Netzwerk- oder Infrastrukturstörung herbeigeführt wurde. Es ist fraglich, ob diese schlechte Performance einen Zusammenhang mit anderen nicht beeinflussbaren Einflussfaktoren hat. Dennoch zeigt sich einerseits, dass die Implementierung im Zusammenhang mit dem Szenario 3 zur Generierung von vergleichsweise grossen PDFs führte. Andererseits sind Aufbau und Layout bei iText und Apache PDFBox mit mehr Rechenleistung verbunden. Apache PDFBox zeigt bei der Last auf dem PaaS-Provider eine höhere Last bei der Ausführung des Szenarios 3. iText wiederum zeigt keine Zunahme, sondern sogar einen Rückgang der CPU-Last.

Bei der Analyse mittels Java visual VM wurde ersichtlich, dass bei der Ausführung von JasperReports viele Objekte angelegt werden, die im Zusammenhang mit den Jasper-Templates stehen. Jedes Feld wird dabei ausgelesen und als Ressource gehalten.

6 Fazit

Aus den erstellten Prototypen und der Evaluation sind viele Erkenntnisse zusammengekommen, die hier noch zusammengefasst werden.

6.1 Implementation

JasperReports

Kleinere Probleme bei der Umsetzung haben die Entwicklung etwas behindert, wie die Probleme mit Bold. Der iReport zeigte das richtig an, aber es wurde nicht in die PDFs übernommen. Dennoch war iReport sehr gut: einfache Handhabung und gute Unterstützung für die Erstellung der Templates. Generierte Reports sind auch als kompilierte Jasper-Files noch verständlich.

Apache PdfBox

Der Zeilenumbruch für einen langen Text muss selber entwickelt werden. Gleicher gilt für die Seitenumbrüche. Für lange Texte ist das eher umständlich. Es fehlt ein High-Level API, darum ist es eher schwierig, Templates zu erstellen, da diese immer auf der Basis von spezifischen Daten-Inputs basieren und eine dynamische Ansicht mit einem Flow-Pattern eher schwierig ist. Das Ausrichten von Textblöcken wird anhand der Seitengröße programmiert, somit ist es auch im Textmodus schwierig, eine Tabelle mit mehrzeiligen Zelleninhalten zu kreieren.

Dennoch bietet sich Apache PDF an, um pixelgenaue Layouts zu erstellen, da die Texte mit Offset-Angaben ausgerichtet werden können.

iText

Es fühlt sich an, als schreibe man ein Worddokument. Die application programming interface ist einfach und klar, Seiten werden automatisch neu generiert. Es benötigt keine Seitenangaben wie Rand und Größe, da Standards diese bereits vorgeben. Ein Standard-Set an Fonts wird als Konstante mitgegeben. Der Hersteller bietet viele Beispiele und hilfreiche

6 Fazit

Blogs an. Die Positionierung einzelner Elemente kann über die Canvas-Funktionen ebenfalls erreicht werden, z.B. beim Definieren von Footer oder Header. iText lässt auch eine dokumentenübergreifende Fonteinstellung definieren, was die Entwicklung erleichtert. Doch auch hier ergeben sich Probleme mit den Fussnoten. Das Dokument müsste erneut bearbeitet werden, um die Anzahl Seiten zu kennen, da diese beim Erstellen noch nicht bekannt sind. Beim Erstellen einer neuen Seite ist die vorhergehende immer um eins kleiner, da beim Erstellen der Seite nicht bekannt war, dass eine weitere Seite dazukommt.

6.2 Performance

6.2.1 Throughput

Der höchste Durchsatz wurde mit ApachePDFBox erreicht, gefolgt von iText und dann Jasper-Reports. Die Analyse des Durchsatzes nach Bytes hat jedoch ergeben, dass dieser fast konstant blieb. Allenfalls ist dies ein Hinweis auf einen Engpass in der Netzwerkanbandbreite.

6.2.2 Latency

Auch bei der Latency by Request ist Apache PDFBox am schnellsten. Die Verarbeitung kann bei den ersten zwei Szenarien unterhalb einer Sekunde durchgeführt werden, was bei JasperReports nicht der Fall ist. iText wiederum hat nur leicht längere Latenzzeiten.

6.2.3 Verfügbarkeit

Keiner der Tests ist jemals ausgefallen, es haben sich auch keine Fehler ereignet, die von JMeter als solche identifiziert wurden. Die meisten Tests haben sich über die Zeitspanne von 10 Stunden erstreckt, dabei haben die Logs ebenfalls keine Ausfälle nachgewiesen.

Dennoch hat JasperReports sehr lange Antwortzeiten, die ebenfalls zu Timeouts führen könnten, was somit die Verfügbarkeit einschränkt.

6.3 Ressources

6.3.1 RAM

Der Verbrauch von RSS-Memory ist sowohl bei iText als auch bei Apache PDFBox meist unterhalb der Memory-Quota. Der SWAP wird einzig von JasperReports genutzt und erklärt wohl die lange Verarbeitungszeit. Der SWAP wird nicht wieder freigegeben, was zu einer Memory-Blase führt.

6.3.2 CPU

Die CPU Last hat nur bei JasperReports durchgängige Spitzen angezeigt, Apache PDFBox hat hier wieder einmal ressourcenschonend gearbeitet. iText hat eine nicht erklärbare tiefe CPU Last im Szenario 3.

6.4 Wer hat nun die beste Performance?

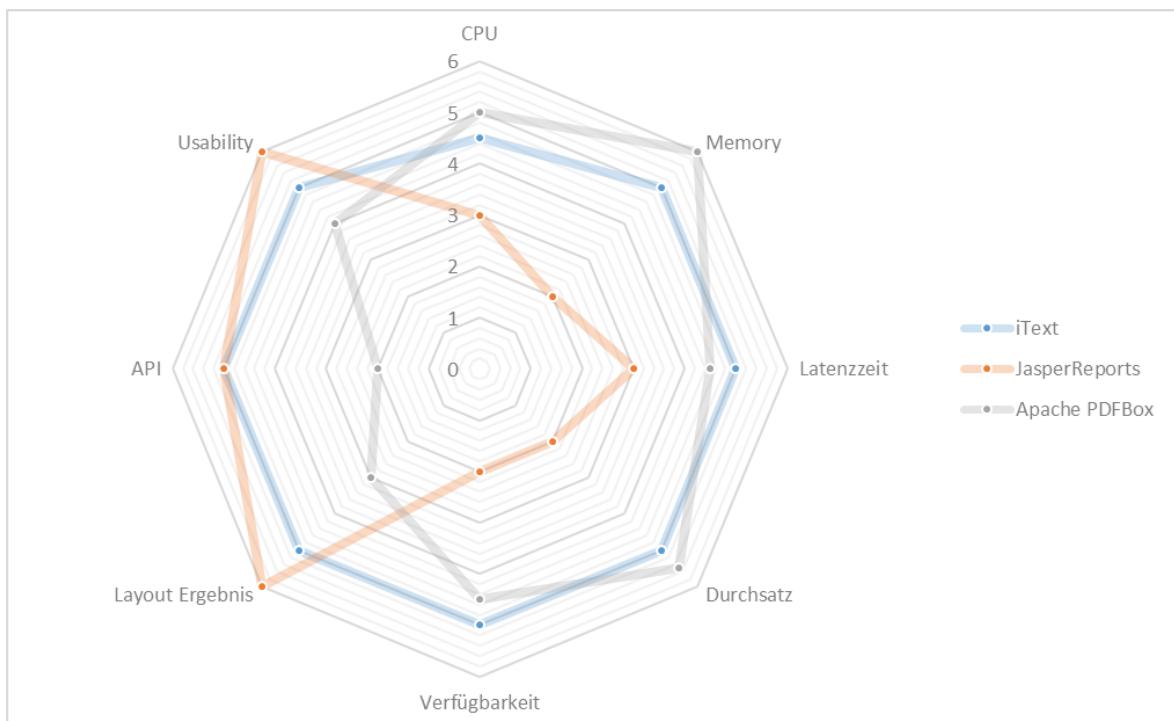


Abbildung 6.1: Netzdiagramm nach Metrik

Würde man für diese Frage einzig und alleine die Daten des Durchsatzes nehmen, müsste man sagen, es sei im Durchschnitt Apache PDFBox. Trotz des Szenarios 3 erreicht dieser immer noch durchschnittlich 70 Anfragen pro Sekunde, gefolgt von iText mit 30 Anfragen pro Sekunde und dann JasperReports mit 14.5 Anfragen pro Sekunde.

Das 95% Perzentil sagt wiederum, dass JasperReports der schnellste ist und das nur, weil das Szenario 3 die Antwortzeiten in die Höhe getrieben hat.

Pauschal lässt sich sagen, dass sich mit Apache PDFBox die höchste Performanz, Ressourcennutzung und Antwortzeiten erreichen lassen. Dennoch haben auch die anderen Seiten, die in verschiedenen Anwendungsfällen besser einzusetzen sind. In der Abbildung 6.1 lassen sich die Tendenzen der jeweiligen erkennen.

6 Fazit

6.4.1 Apache PDFBox

Diese hat klare Stärken, was die technische Performance angeht. Die Umsetzung ist klar auch historisch geprägt. Es ist ein Tool primär zur Extraktion von Daten und nicht für die Erstellung von Reports gedacht, was es auch so performant macht. Das geeignete Einsatzgebiet im Umfeld von Reporterstellung liegt auf dem Gebiet von statischen Layouts wie Flugtickets oder Konzertkarten.

6.4.2 iText

iText wurde genau dafür kreiert, um Reports zu generieren. Dies ist klar erkennbar an dem versatilen und umfangreichen API. Es beherrscht auch den komplexeren PDF-Aufbau ohne Performanceverlust. Für einen Online-Einsatz wird es als brauchbar empfunden. Das API ist gut genug, um Reports leicht verständlich zu codieren, und bleibt darum wartbar.

6.4.3 JasperReports

Es gibt klare Vorteile bei visuell aufbereiteten Reports, leider hat aber dieser Vorteil keine nutzbare Performance für eine Online-Verarbeitung gezeigt. Dies bedeutet: JasperReports ist für einen Einsatz in einem Microservice nur sehr bedingt geeignet. Ein geeignetes Einsatzgebiet wären zeitlich klar definierte Reportgenerierungen. Oder auch wenn komplexe Reports womöglich von Nicht-Entwicklern definiert werden sollen, ist iReport dafür ein geeignetes Hilfsmittel.

6.5 Ausblick

Aus den Performance-Messungen entstanden einige Fragen im Zusammenhang mit der Implementation, die noch zu adressieren ist, wie z.B.:

- Sind Engpässe in Applikation oder Netzwerk Ursachen für die schlechte Performance?
- Sind Fehler in der Applikation, die einen unnötigen Programmcode ausführen?
- Sind Einstellungen möglich, um Zugriffe auf Dateien wie Bilder zu verbessern?

Und in dieser Arbeit nicht addressierbare Fragen bezüglich Skalierbarkeit der PaaS.

- Mit welcher Anzahl Server können X Anzahl User performant bedient werden?
- Wie verhält sich das Auto-Scaling?
- Wie gut funktioniert das Load Balancing ?

Diese Fragen könnten in einer weiteren Arbeit vertieft werden.

Glossar

Dyno Dyno ist ein Webserver bei der PaaS Heroku. 3

JSON JavaScript Object Notation, ein Datenaustauschformat. 8

OSRE Open Source Reporting Engine, Framework für das Verarbeiten und Erstellen von Reports. I, 1, 30, 35

POST Eine der HTTP-Anfragemethoden. 8

RAM Random-Access Memory bezeichnet den Arbeitsspeicher bei Computern. 4

SOA Service Oriented Architecture, beschreibt den Grundgedanken von verteilten Systemen herunter gebrochen auf einzelne Services. 1

Akronyme

API application programming interface. 33, 36

CLI command-line interface. 16

CPU central processing unit. 28

FOSS Free Open Source Software. 5

HTTP Hypertext Transfer Protocol. 7

IaaS Infrastructure as a Service. 1, 1

PaaS Platform as a Service. 1, 3, 4, 28

PDF Portable Document Format. 7

REST Representational State Transfer. 7

VU virtuelle User. 29, 30

Abbildungsverzeichnis

3.1	UML PDFEngine Implementierungen	10
3.2	UML Szenario Implementierungen	11
4.1	Evaluationsablauf	15
4.2	PDF-Resultate Szenario 1 (v.l.n.r iText, JasperReports, Apache PDFBox) .	18
4.3	PDF-Resultate Szenario 2 (v.l.n.r iText, JasperReports, Apache PDFBox) .	19
4.4	PDF-Resultate Szenario 3 (v.l.n.r iText, JasperReports, Apache PDFBox) .	20
5.1	Vergleich - Durchsatz nach Szenario	23
5.2	Durchsatz in Bytes	24
5.3	Latenzzeit im Testzyklus	25
5.4	Latenzzeit nach Szenario und OSRE	26
5.5	Fehlerrate nach Latenzzeit über 2000ms	27
5.6	Memoryverbrauch Testlauf	28
5.7	Vergleich - Durchsatz nach Szenario	29
5.8	Vergleich Baseline mit grösserer Datenmengen	30
5.9	Vergleich Baseline mit mehr virtuellen Usern	31
6.1	Netzdiagramm nach Metrik	35

Literaturverzeichnis

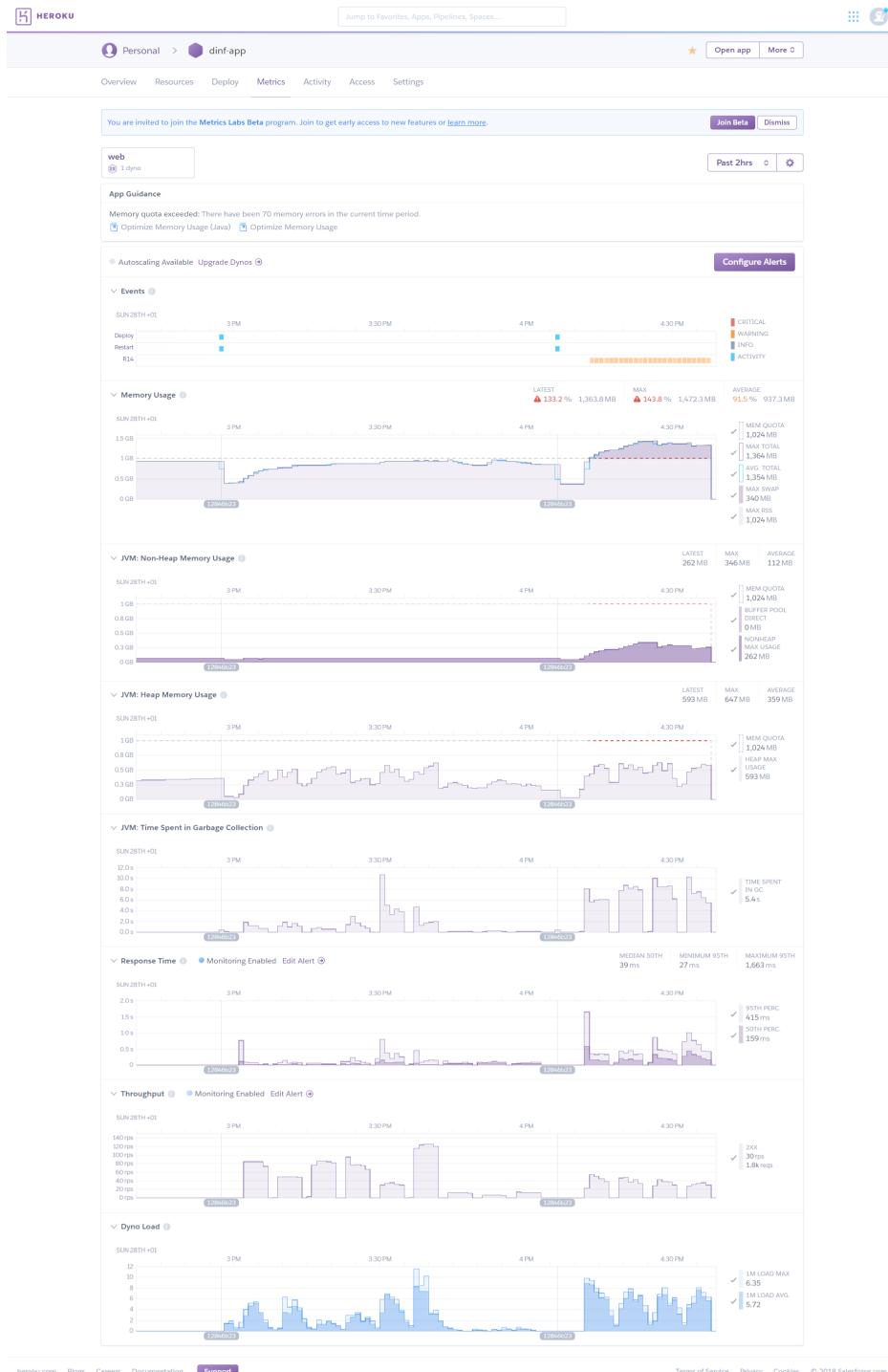
- [1] I. Molyneaux, The Art of Application Performance Testing: From Strategy to Tools, ser. Theory in practice. O'Reilly Media, Incorporated, 2014. [Online]. Available: <https://books.google.ch/books?id=IKXboAEACAAJ>
- [2] I. Sommerville, Software engineering. Pearson, 2012.
- [3] J. Whitington, PDF explained. O'Reilly, 2012.
- [4] D. R. Heffelfinger, JasperReports 3.5 for Java developers: create, design, format and export reports with the world's most popular Java reporting library. Packt Publ., 2009.
- [5] B. Lowagie, iText in Action, Second Edition. Manning Publications, 2010.
- [6] "Apache pdfbox and fontbox 1.0.0 released," Feb 2010. [Online]. Available: <http://www.h-online.com/open/news/item/Apache-PDFBox-and-FontBox-1-0-0-released-932436.html>
- [7] R. B. Miller, "Response time in man-computer conversational transactions," Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS 68 (Fall, part I), vol. 33, p. 267-276, 1968.

Selbstständigkeitserklärung

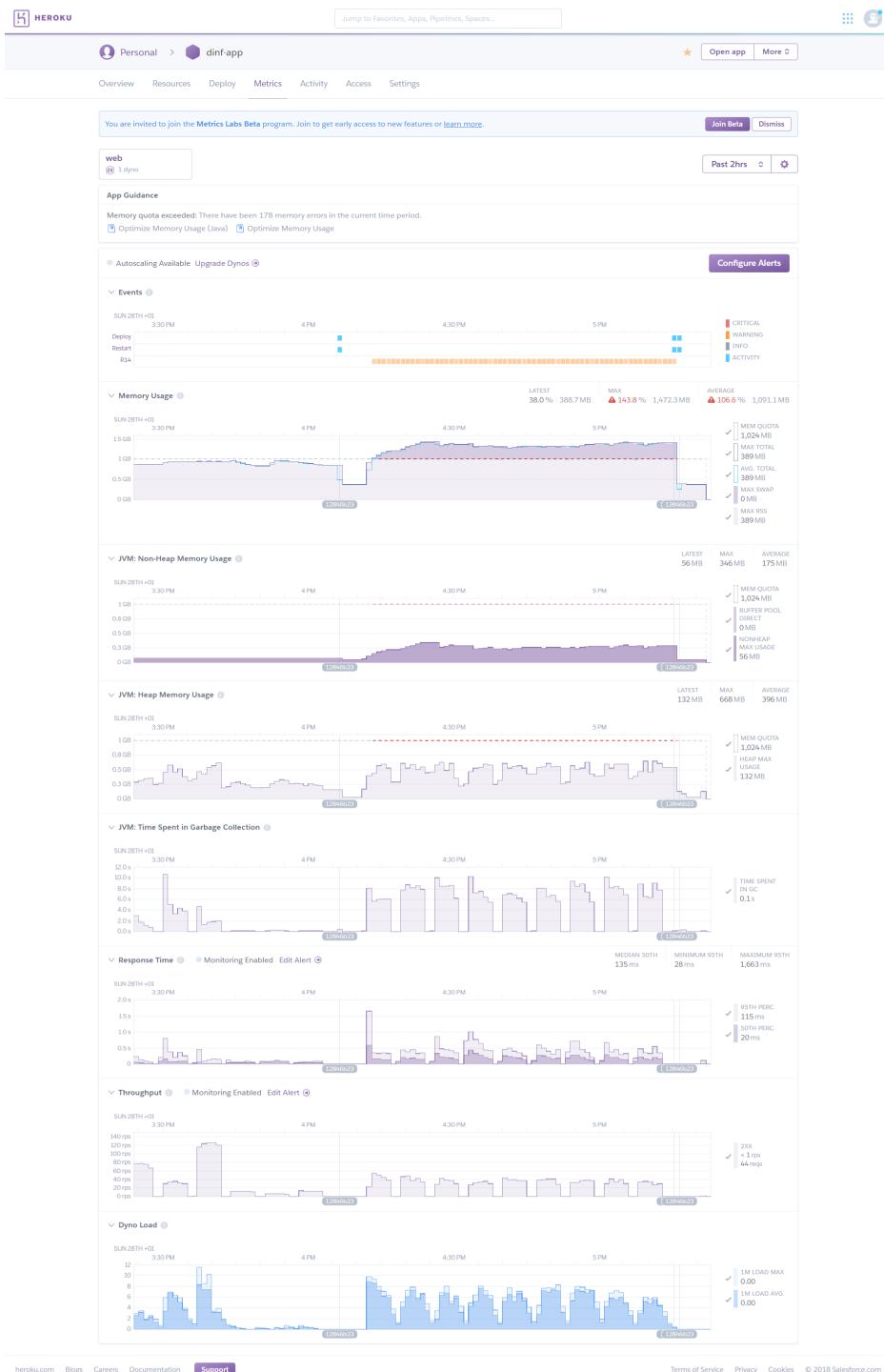
Ich erkläre hiermit, dass ich diese Thesis selbstständig verfasst und keine andern als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Ich versichere zudem, dass ich bisher noch keine wissenschaftliche Arbeit mit gleichem oder ähnlichem Inhalt an der Fernfachhochschule Schweiz oder an einer anderen Hochschule eingereicht habe. Mir ist bekannt, dass andernfalls die Fernfachhochschule Schweiz zum Entzug des aufgrund dieser Thesis verliehenen Titels berechtigt ist.

Ort, Datum, Unterschrift

Anhang



Literaturverzeichnis



Literaturverzeichnis

