Designing Infrastructure for an Online Teaching Assistant

Bachelor project

Department of Computer Science at the University of Copenhagen
(DIKU)

Truls Asheim <truls@asheim.dk>

June 21, 2015

Contents

Chapter 1

Introduction

In this report, we will describe the design and implementation of an infrastructure for a system facilitating submission and automated correction of student assignments.

Unfortunately, some of the learning objectives proved overly ambitious for the time frame that we had available. In particular, we had hoped that our proof of concept implementation would be in a much more complete than it currently is. This impairs our ability to present firm conclusions but we do, however, expect that the design and testing methodologies described provides a solid starting point for such conclusions to be drawn at a later point.

A Master's project carried out in the spring of 2014 has laid the foundation for this work by discussing the intricate aspects of performing automated assessment of assignments[1]. This work builds upon the previous project by describing an infrastructure for a complete system supporting the submission and automated assessment of assignments

FiXme Note: Rewrite last part

1.1 Background and motivation

As a result of the study progress reform recently passed by the Danish parliament, students are required to finish their university studies faster than previously[2]. A consequence of this, is that many students can no longer find the time to act as TA's alongside their studies.

This has sparked a general interest at DIKU[1] to investigate implementing an automated assignment assessment system in order to reduce the workload for teachers and teaching assistants TA's.

Since the assignments of many Computer Science courses require students to complete programming homework, Computer Science finds itself in a unique position with regard to take advantage of automated assignment assessment systems.

[1] Department of Computer Science, University of Copenhagen

The system that we here will describe does not intend to replace TA's entirely, but rather automating the tedious of compiling an executing student programs. Human involvement is still needed to asses submissions as a whole.

An additional goal of this systeom is to be used as an Intelligent Tutoring System (ITS)[3] which can provide support for students while they are solving their assignments.

1.2 Related work

Tools for performing automated assessment of programming assignments is currently an actively researched topic. As a result of this, a large number of systems for performing this task has emerged over the years [4].

Chapter 2

Analysis

We will describe some of the aspects of handling student submissions in this analysis chapter of the report

2.1 Data overview

The system are required to handle several sensitive data items which must be protected from adversaries before the online assessment system can be put into production. The types of data are listed here in order of decreasing sensitivity.

1. The submissions of other students
2. The code for assessing an assignment
3. The completed assessments of assignments.

The first two items on the list are close in terms of sensitivity. We regard the submissions of other students as being the most sensitive data since they have a larger potential to enable plagiarism.

Since it would have less impact if an adversary gained unintended access to the code for assessing an assignment we don't assign it the same sensitivity. Getting access to the assessment code would only tell you what kind of output that your assignment should produce in order to pass. It would not tell you how to make the actual assignment. Thus, an assignment that is "‘tuned"’ to produce the output or contain the phrases that is checked for would not pass subsequent human inspection. A possible exception to this occurs should this system ever be used to correct assignments with a single correct answer. In this case, getting hold of the assessment code equates acquiring the answers for the assignment

Assessments generated by the system is the least sensitive data being handled since the, a) do not identify students by name and b) TODO

FiXme Note: Is it always the case that they do not identify students by name?

Because of the data that we handle, securing assignments from unauthorized access is our primary priority.

2.2 Personas

2.2.1 Student

TODO: We assume that students can not be trusted

2.2.2 Teacher

TODO: We assume that teachers can be trusted

2.3 Assignment types

Types of assignment. Some assignments are complete folders, some assingments are just simple files. Some students may prefer to work in a straight forward web browser instead of uploading files.

```
There are also various types of assignment. Covering this would better
introduce the next section where you talk about the various phases
that there can be to the assessment of a submission.

Different assignment types:

    Submit a byte stream, e.g. input from a multiple-choice interface (including free-fo
    Submit a file.
    Submit an archive.

For each of those we may want to:

    Validate the structure of the submission.
    Check that they are not cheating or trying to game the system.
    Interpret some part of their submission as code and run it.
    Check that the environment they leave behind after running is valid (e.g. produce a
```

2.4 Assessment flow

A submission goes through several phases of assessment categorized as either static or dynamic assessment.

The static testing phase assess the static properties of submissions. In this phase, submitted code should never be executed

The dynamic testing phase will actually execute student submitted and compare its responses against a set of included reference answers.

2.4.1 Rationale

The reason for this split is that we can let two different security models govern the two phases.

In the static phase, evaluation programs are given complete access to the submitted source code. There is no technical limit in this phase preventing the testing scripts written by teachers from being executed. Furthermore, accidental student code invocation can also arise from vulnerabilities in utilities that is invoked as part of the assessment process. An example of such a vulnerability is the buffer overflow that was triggered in the `strings` utility by using it on certain ELF files [5].

FiXme Note: this makes no sense

Since this phase is sandboxed, no risk is posed to the wider system if malicious student submitted code is inadvertently run. but assess

In the dynamic phase both the student submitted code and the assessment code is run, but in two different sandboxes with the only interaction between them being pipes. This isolates the assessment code and

2.5 Submission lifecycle

In this section we describe the life cycle phases which an assignment goes through from submission to the assessment being returned to the student. The life cycle of a specific assignment depends on several factors. Particularly if we are dealing with an identified oor anonymous submission

Common for both types of assignments is that they are assigned a Universally Uniqueue ID (UUID) upon submission which will be used to identify the assignment for the remainder of its lifecycle.

Along with being able to uniquely identify a submission, we also need to

We could do this by associating a set of metadata items to the submission.

An anonymous submission will only exist on the server while the assessment is taking place.

An attributed submission, on the other hand, will persist on the server for a certain amount of time. This, for instance, could allow for repeated submissions to be differentially encoded providing bandwidth savings.

2.6 Executing untrusted code

If the OnlienTA system is to fulfill its intended role, it is a necessity that it executes arbitrary code submitted by students. . For obvious reasons, we cannot trust student submitted code not to perform inappropriate actions, may these be unintended side-effects of programming errors or deliberate malicious attempts to exploit the system.

FiXme Note: rewrite

2.6.1 Sandboxing

We will handle this challenge by executing the student-submitted code in sandboxes where they have no chance of inducing side-effects on other parts of the system, gaining access to privileged data or affecting the assessment of other students assignments.

Sandboxing technologies range from fully-flgedged virtual machines to rule-based execution environments for programs. We will here focus on containers which is a type of OS-level virtualization and resource confinement which by enforcing separate namespaces for e.g. networking, filesystems and system permissions. The Linux kernel provide support or this kind of sandboxing through the feature sets `namespaces`[6], cgroups, rlimit and seccomp.

There are several well-known "‘packaged"’ implementations OS-level virtualization for Linux which is implemented using the namespaces features. In this section we will describe a couple of these that has been considered for use with OnlineTA.

Docker

Docker is a relatively recent addition to the family of application containerization tools. Opposed to most other OS-level virtualization technologies it focuses on application isolation rather virtualization of an entire operating system. It has popularized a new way of deploying and executing applications where, instead of applications being installed on a shared system, applications are deployed by starting a container which includes the application, its dependencies and its configuration. [7]

Docker comes with a complete set of tools for building and managing containers. These could be useful in supporting the TA's job of developing and testing assessment units for assignments.

On the other hand, the codebase of Docker is very large and contains a lot of features that are superfluous for our use case. A large codebase also has a higher chance of containing security critical bugs. The widespread adoption of docker for security critical applications, however, mitigates this concern somewhat.

Priviledge escalation vulerabilities has historically been more likely to arise from peripheral feature sets (as seen in the recent QEmu floppy driver vulnerability [8]), rather than the core containment functionality itself.

Sandstone

As part of a related student project, a minimalist and modular interface to the Linux container APIs were developed. The philosophy behind this system, named sandstone, is to provide access to the sandboxing features of Linux as small individual building blocks which can be composed, providing the program executed with the desired level of isolation.

The building blocks of Sandstone is implemented as small programs written in the C language, each of which implements a part of the Linux container system.

The programs are composed through command chaining. Exemplifiedied, the composition works as follows: Let a, b, and c be programs in Sandstone and let p be the program that we wish to execute in a sandboxed environment. We execute the process with the command line `a b c p`. Then b will be constrained by the isolation provided by a, c will be constrained by a and b and p will be constrained by a, b and c and will thus be executed with the level of isolation that we desired. [1]

There are good reasons to be wary of custom built security critical utilities. Security is hard to get right and custom built utilities that never see widespread use will never go through the public scrutiny offered to more publicly visible tools. A mitigating feature of Sandstone in this regard is its relative simplicity and the limited amount of code used in its implementation.

Conclusions

The containers each have their own individual strength and weaknesses and both of them are able to serve the purpose for containing the assessment process in OnlineTA. However, further practical testing is needed draw firm conclusions about their suitability. Such testing is enabled by the fact that our design, that we will later present, features support for swapping containerization method.

2.7 Design principles

We have devised a number of principles that our system design will be based on. These principles aims to simplify the design of our s

Modularity We want the components of our system to be modular and loosely coupled. Large monolithic components tends to foster unmanageable complexity which makes them harder to maintain and review for security problems. Modularity also adds flexibility to our system by enabling certain components to be swapped.

Clearly defined minimal interfaces

Minimum permssions A common "‘best practice"’ related to the security aspects of software development is to assign a minimalized set of permission to a component required by the work that it needs to perform. Traditional Unix environments only makes the distinction between a superuser (root) and a regular user. It’s common for applications to be started with full superuser permissions and then, after performing privileged actions,resume operations as a unprivileged user. It’s important to make sure that a process can never regain the privileges that it has dropped.

The sharp separation between superusers and regular users no longer exists in model versions of the Linux kernel following the introduction of the *Capabilities* interface [9].

Capabilities allows individual processes t

Data ownership separation The most basic permission management primitive in Unix operating systems is the concept of a user. We wish to enforce fine grained user separation across the system such that a given user only have access to what is absolutely necessary. A common concern of executing applications inside a container is what happens in the unlikely event that a process successfully manages to escape the container. In this case, the attacker would usually end up having the same permissions as the process executing the container. Therefore, we can mitigate the consequences of a potential container escape by making sure that the user executing container has access to as few things as possible in the wider system.

2.7.1 Securing special capabilities

Despite our best efforts, we can never completely eliminate the possibility that an adversary will be able to take advantage of the capabilities given to our programs and force them to preform unintended actions. For instance, gaining arbitrary code execution in a process with the `chown` capability would allow one to change permissions of any file in the system. Since our security model to a large extent relies on users having access to a minimalized set of files, we would consider this to be a rather serious security incident.

FiXme Note: That's also because our programs can't always reduce their privileges (they will need them for later on), so we need to make sure that we both give sufficient, but also not overly lavish capabilities.

Secure Computing (seccomp) [10] is a facility provided by the Linux kernel to provide application sandboxing through the filtering of syscalls. It allows an application to unidirectionally transition into a secure state where all of its syscalls are filtered by specified rules. Building on our example in the previous paragraph, we can use the seccomp to filter the `chown` syscall such that it only allows changing the owner of files to a UID within an allowed range, e.g. $(\text{aaa000})_{36} - ((zzz999)_{36})$ and is only allowed to change ownership files within the submissions directory.

We can use seccomp to build an extra layer of security around our programs which limits the effects of

FiXme Note: discuss problem of go spawning multiple threads per default somewhere

2.8 Handling user identities

We will support assessing submissions from two groups of user: anonymous and authenticated users. Submissions from these groups will be handled in two different ways, both with regard to what services are offered to the user, but also how we represent their submissions internally. Submissions made by an authenticated user will be referred to as an *attributed* submission.

Anonymous submissions are removed after their life cycle has concluded. Only the assessment of the submission persists until it has been read by the submitting user.

FiXme Note: We should hold off describing our implementation of handling usernames through

Identified submissions

For the time being, the scope of OnlineTA is limited to students at DIKU. Thus, we can take advantage of the standardized usernames assigned to everyone who is in some way associated with the University of Copenhagen (KU-usernames), both students and employees. KU-usernames are matched by the regular expression `ˆ[a-z]{3}[0-9]{3}$`, with the exception that vowels aren't used as letters in generated usernames. We can use this username generation scheme to our advantage since we can interpret the KU-usernames as base 36 numbers and thus achieve a stateless and bidirectional mapping between Linux UIDs and KU-usernames, e.g.:

$$(\text{gfr534})_{36} = (993919360)_{10}$$

Since UIDs in Linux are represented as a 32-bit signed integer, actual implementations of this scheme needs to take into account that the (numerically) largest valid KU-username $(zzz999)_{36}$ is larger than $(2^{31})_{10}$.

Verifying user identity Another ongoing student project are designing and implementing a purpose-built GPG key server which are intended to be implemented into the OnlineTA system. The purpose of this key server is to provide a mapping between a KU-id and a GPG-key. Since all identified submissions are

At this stage, we have not found a suitable model for supporting group submissions

2.9 Interacting with the system

In this section, we describe modes of interaction with the system for making submissions and receiving assessments.

Since this system is intended to be used even by freshmen submitting their first assignment, providing a simple, familiar, universal and immediately accessible interface for submitting assignments is an important goal. However, a secondary purpose of this system is, not just aiding TAs in their grading work, but also to provide students with continuous feedback while they are solving their assignments. It's therefore important that we meet students where they are and provide them with (from their perspective) the most seamless interaction possible.

The first goal is best achieved through a well-designed web interface for uploading assignments. The interface should, when possible, provide immediate feedback within the browser. As previously mentioned, we have based our user identification on verifying that an assignment has been signed with the GPG-key belonging to the user claiming to have submitted the assignment. This poses a potential obstacle when implementing authenticated submissions through the web interface. One option, is to require students to GPG sign their assignments prior to submission.

FiXme Note: Talk about when system overload causes long submission times

Many additional submission interfaces has been proposed since the conception of OnlineTA. The prevailing proposal has been to implement a command line interface which would allow a student to submit the current folder or a file for assessment. The Windows equivalent of this would be a Shell Extension adding tantamount functionality to the right click menu of files and folders. We are not qualified to provide equivalents for other platforms such as OSX, but in that particular case, the command line interface would also be feasible. A tangential advantage of this submission method is that it could be implemented to only transmit deltas between repeated submissions and thus significantly reduce upload time and internet bandwidth required for submission.

Another submission interface under consideration is Git. Since revision control systems (primarily git) are already an integrated part of the assignment solving workflow of many students, we consider eventual addition of git an important goal. Assignment

FiXme Note: TODO: Argue why git is nice

Chapter 3

System Architecture Design

In this chapter, we propose a design for our system. We will use the principles that we laid down in the previous chapter and present them as a part of an integrated system

3.1 General concepts and overall goal

With security as the, as previously stated, principal goal, we focus our design around several small processes, each with a well defined purpose and interactions. We furthermore enforce a principle of minimum permissions across our entire system.

3.1.1 Container support

The design of the system enables the containerization model to be swapped, such that different containers can be used for the assessment of different assignments. Supporting this modularity is important since different assignments and teachers may have varying demands of the container used for assessment.

3.2 Users

There are two levels of users in the OnlineTA system. One is the user owning the daemons responsible for handling the main assessment flow, and the other is the users owning the submissions. In the case of anonymous submissions

3.2.1 Representing users on system

Name service lookup services in GNU/Linux systems are provided by the Name Service Switch (NSS) subsystem which is implemented in the GNU C Library (glibc). Lookup databases include users, groups and DNS names. The NSS system provides a unified interface for performing lookups in a database backed by multiple services. The NSS system is extensible through services which is called in the order specified by the /etc/nssswitch.conf file. In the OnlineTA system, we add a password mapping module which can be used for looking up user names to UIDs and vice-verca[11].

This allows us to represent our dynamically created users on the Linux system and thus avoid having to define them statically in the /etc/passwd file.

We perform user handling in two separate NSS modules for handling named an anonymous submission respectively. The usernames of identified submissions are assigned by an NSS module which implements the bidirectional UID to KU-username mapping previously described. conversion scheme previously described.

We handle the usernames of anonymous submissions slightly differently through a different NSS module. This module works on lists of preallocated users and is autonomously responsible for maintaining lists of allocated and non-allocated users

FiXme Note: rewrite

3.3 Storage

In our current design, availability of persistent local disk storage is assumed. The directory structure used for storing submission data is as follows (figure 3.1):

`submissions` The parent of the directory tree

`<UUID>` A directory named after the UUID assigned to a submission is placed as a sub directory of the submissions directory. It holds all the files related to a submission, including it's assessment. he contents of this directory are owned by the user who made the submission and thus it can only be accessed by an assessment unit running as this user.

`incoming` This directory is where the metadata files of all newly created directories are stored. It is monitored by the scheduler using `inotify` which, after a metadata file is scheduled it is moved away from this directory

`accepted` This directory holds metadata files which has been accepted for processing. The separation of incoming and accepted metadata files is needed due to limitations of inotify

FiXme Note: elaborate

3.4 Capabilities

In section 2.7 we introduced the concept of capabilities. In this section, we will describe the properties[9] of the specific capabilities that are referred to henceforth.

`CAP_CHOWN` The process in possession of this capability can use the `chown` syscall to change the owner of any file or folder on the system.

`CAP_SUID` Allows a process to freely change its effective UID.

```
     submissions
        /|\
       / | \
      /  |  +-----+------------+
     /   |        |      |     |
 UUID1   UUID2   UUID3   |  incoming
                         |     /\
       ------------------+    /  \
      |                      /    \
      |        UUID1.metadata UUID2.metadata
   accepted
      |
UUID3.metadata
```

Figure 3.1 – Tree showing our folder structure in a state where one assignment (UUID3) has been accepted an scheduled, while two assignments (UUID1 and UUID2) are waiting to be picked up by the scheduler

`CAP_SETPCAP` This capability allows a process to either add a capability from the bounding set of capabilities inherited from its calling process or drop a capability from its current set.

`CAP_SYSADM` This capability allows a process to perform a large number of administrative tasks. We are primarily interested in using the `mount` syscall to prepare the filesystems used for assessment.

3.5 Components

Our proposed design for the OnlineTA system is split onto a number of different components which are described in this section. Their relationship is shown in figure 3.2.

Frontend The frontend is responsible for accepting submissions. In case of attributed submissions it is also responsible for verifying the identity of the user submitting the assignment. It will also perform the initial storage of the submission. Furthermore, the frontend is responsible for returning the assessment to the student.

The frontend has the following roles

- Accepting submissions from the HTTP proxy
- Verify GPG signature of assignment to verify submission attribution
- Generating and assigning a UUID to the submission

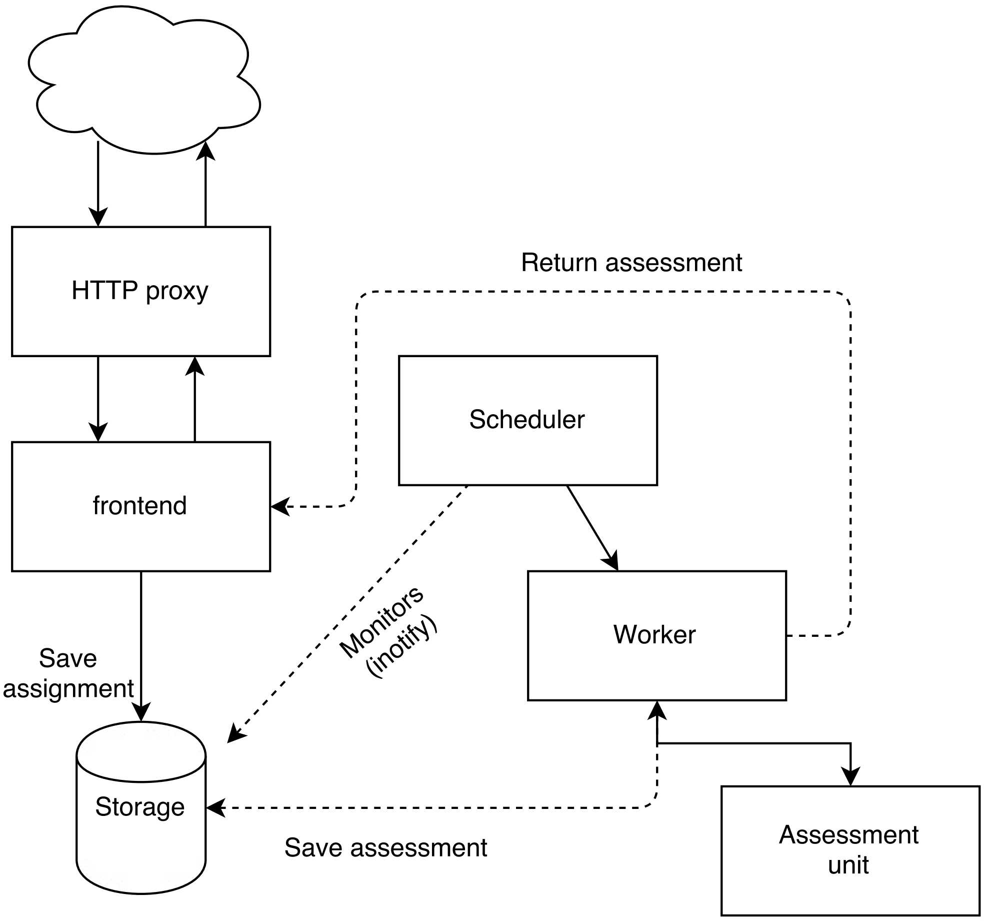

Figure 3.2 – Flowchart showing interactions between the components of the OnlineTA system

- Writing the initial metadata file
- Creating directory for containing the submission and setting ownership of files

This component require the `CAP_CHOWN` capability and write access to the submission root folder.

FiXme Note: It would be nice to state that the "work queue" is just a folder of folders, each owned by the respective user that made the submission.

Scheduler The scheduler will pick up the assignment after it has been saved to disk and pair it with a suitable assessment unit. Assignment scheduling is based primarily on a priority assigned to the submission by the frontend. Submissions with equal priorities are scheduled on a FIFO basis. If no priority is provided by the assignment, it could assign a priority based on a number of factors, e.g. time remaining before submission deadline. The scheduler directly spawns processes for executing assessment units.

The scheduler has the following roles:

- Monitoring the incoming directory for new submissions
- Pairing submissions with an appropriate assessment unit
- Scheduling and prioritizing assessment of assignments
- Invoking workers for performing the actual assignment

The scheduler does not need any special capabilities, but it requires read and write permissions to the metadata queue directory.

Worker A worker process is spawned by the scheduler and its overall purpose is to set up and initiate the assessment process. It is initially started as the `onlineta` user but after the assessment environment has been configured and the containers has been created it will `suid` to the user owning the submission and thereby shed its special capabilities.

Specifically, its roles are the following

- Setting up the environment for performing an assessment including mounting filesystems
- Setting up containers for assessment
- Relaying assessment to frontend

The special capabilities required by the Worker process are `CAP_SYSADM`, `CAP_SUID` and `CAP_SETPCAP`

Assessment units An assessment unit handles the actual assessment of an assignment.

3.6 Interactions

The scope of the current implementation is limited to supporting direct interactions via a web browser, however, the additional modes of interaction previously described can be implemented on top of the interface that we will describe in this section.

In the current design, all interactions with the system occurs over a secure HTTP connection. We use the first part of the request URI to determine which action is requested by the user while subsequent URI components are used as parameters for that request. The system supports the following actions:

/ This URI can be issued as a HTTP GET request and will return the main index page which will contain a list of courses and assignments availiable on that server

`/submit/<course>/<assignment>/` When a HTTP POST request is issued to this URL it will be handled as an anonymous submission. If the submission is accepted, the fronted will repsond with a page showing the assigned submission UUID. This UUID can then be used in order to retrieve assessments once they are completed

`/submit/<course>/<assignment>/<KU-username>` This request type is the same as the previous except that it will perform an attributed submission. Thus, an assignment submitted through this request will only be accepted if all of its constituents are signed by the GPG-key mapped to the proclaimed user.

`/query/<uuid>` Used to query assessment status of assignment identified by given UUID. This request is site-effect free and can thus never alter the state of the system. The status of an assignment can be the following

ACCEPTED The assignment has been accepted and is awaiting schediling

QUEUED The assignment has been picked up by the scheduler and is awaiting prioirization

PROCESSING The assignment is in the process of being assessed

FINISHED Assessment has been successfully completed

FAILED Assessment has failed

`/assessment/<uuid>` Retrieves the assessment of a submission if available. This request is not nullipotent since the assessments of anonymous submissions will cease to be available once this request has been performed.

3.6.1 Relaying assessment to user

TODO: Describe what happens when a user requests an assessment.

3.7 Large Submissions

Some courses may require large submissions which could saturate any feasible internet link through only a limited number of concurrent uploads. We propose that the upload bandwidth requirement can be reduced by applying a delta transfer algorithm on repeated submissions.

The rsync algorithm is an abvious candidate for solving this problem. However, the alogrim assumes that the receiving end has access to all files. Honoring our porperty of minimal user permissions the submission receiving process won't have access to the files of the previous submissions. We therefore propose a variation of the rsync algorithm built for our purpose. Since we cannot rely on being able to retrieve the state of the remote files we must be able to keep the difference between local and remote files locally. TODO

TODO: Finish and compare to git (Why cant git perform the differential transfer?

Chapter 4

Measurements

A goal of this project is to determine the servber capacity required by the OnlineTA system. Since the current implementation is not yet at a state where we can reliably perform such measurements, we will instead focus om describing measurements which can be performed of the final implementation in order to assess server capacity requirements.

Two of the metrics that we will use to judge the outcome of our benchmarks are the following:

Assessment latency describes the delay between the submission of an assignment until the assessment is ready

Assessment throughput describes how many assessments that can be produced within a given time frame.

4.1 Purpose

Benchmarking this system has two purposes. The first is to determine whether the "‘baseline"’ i.e., best possible, performance of the system is acceptable. The second is to determine how many simultaneous assessments we can perform without the assessment throughput dropping to an unacceptable level.

Based on these two measurements we can a) determine if the architecture of the system is fast enough and b) the hardware requirements for scaling our system to the required level. Additionally, we can determine which hardware components are optimal for our purpose, e.g., if disk I/O proves to be a limiting factor we should use production hardware featuring Solid State Drives (SSDs)

4.2 Benchmark setup

Benchmarking should ideally be performed in an environment of the highest possible control and predictability in order to eliminate sources of error and achieve consistency in our benchmarks. In our case, upholding these ideals means that we should use two separate machines for performing the benchmarks connected through a high-bandwidth, low-latency LAN link

Since interaction with our system occurs over HTTP we can base our benchmarking setup on one of the large number of HTTP benchmarking tools available such as ApacheBench (ab)[1] or wrk [2].

During benchmarking, we need to continuously monitor system statistics in order to identify the bottlenecks limiting overall performance. A suitable utility for this purpose is `sar(1)` of the sysstat[3] project which is able to collect and log a huge number of Linux system performance metrics.

4.3 Assignment types

The resource needs for an individual assignment differs with the type of assignment being assessed. One assignment might require significant mounts of CPU computational time while another might demand hefty amounts of memory. In order to claim complete testing coverage it is therefore important that we make sure that we include many different kinds of assignments in our tests, preferably in interlacing order

For the most realistic results, old assignments from previous course runs are to be preferred.

4.4 Acceptance thresholds

In order to prevent making overestimated hosting recommendations it's important to establish acceptable thresholds for assessment throughput. A system that is scaled toward delivering assessments "'as fast as possible"' under any circumstance will have significantly different requirements than a system where we accept queuing of assignments and thus delaying assessments during peak load.

The perceived usefulness of the system and overall student satisfaction will depend greatly on striking the right balance between acceptable assessment throughput thresholds and cost of hardware deployments.

4.5 System scaling requirements

As our initial scope is limited to performing assessment of the assignments of DIKU coursess, we can develop an idea of how large our assignment throughput needs to be fairly easily. E.g if we have 100 students enrolled in the introductory programming course, we can calculate an upper bound on the amount of resources consumed. This can be done by calculating the expected resource requirements for each assignment in the course. Our required throughput is then

[1] `http://httpd.apache.org/docs/2.2/programs/ab.html`

[2] `https://github.com/wg/wrk`

[3] `http://sebastien.godard.pagesperso-orange.fr/`

4.6 Benchmarks

4.6.1 Assessment latency

We test the base assessment latency by simply measuring the time required to assess

4.6.2 Assessment throughput

In this benchmark, we simply increase the number of concurrent assignments being assessed until the number assessments per minute drops below the acceptable threshold.

As long as sufficient hardware resources are available, we expect the assessment throughput to scale linearly with the induced degree of assessment concurrency. However, once a hardware resource starts acting as a limiting factor, we expect to see decreasing performance. Our system metrics logging can then be used to determine bottleneck and thus allow us to focus on enhancing system performance in this area.

In order to perform a complete evaluation of the system, we need to perform a th

We will perform benchmarks such as

- "‘baseline assessment latency"’ "= How long does it take for an "‘empty"’ assignment to pass through the assessment system. - What is the maximum number of assignments that we can handle in parallel i.e. when does throughput and/or latency worsen - Benchmarking assignments which has their weight on various resource requirements. such as one assignment requiring a lot of memory and another being heavy on disk io.

- Edge cases such as large number of small assignments, small number of huge assignments.

Chapter 5

Discussion and conclusions

We will end this report by presenting our conclusions and discussing the wider implications of our proposed design.

5.1 Practical deployment considerations

In this section, we will devote some space to consider the practical and economical implications of the possible hosting choices that we have for our design.

It may seem that a cloud hosting provider such as Amazon or Digital Ocean would provide the ideal environment for hosting OnlineTA,

Bibliography

[1] Oleksandr Sturmov. “Online TA”. Master Project. 2014. URL: https://github.com/oleks/onlineta/raw/master/report/output/onlineta.pdf (cit. on pp. 3, 9).

[2] University of Copenhagen. *Fremdirftsreform (in danish)*. 2014. URL: http://nyheder.ku.dk/fremdriftsreform/ (visited on 06/19/2015) (cit. on p. 3).

[3] Reva Freedman, Syed S. Ali, and Susan McRoy. “Links: What is an Intelligent Tutoring System?” In: *Intelligence* 11.3 (Sept. 2000), pp. 15–16. ISSN: 1523-8822. DOI: 10.1145/350752.350756. URL: http://doi.acm.org/10.1145/350752.350756 (cit. on p. 4).

[4] Petri Ihantola et al. “Review of Recent Systems for Automatic Assessment of Programming Assignments”. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Koli Calling ’10. Koli, Finland: ACM, 2010, pp. 86–93. ISBN: 978-1-4503-0520-4. DOI: 10.1145/1930464.1930480. URL: http://doi.acm.org/10.1145/1930464.1930480 (cit. on p. 4).

[5] Murray McAllister. Oct. 26, 2014. URL: https://bugzilla.redhat.com/show_bug.cgi?id=1157276 (visited on 06/21/2015) (cit. on p. 7).

[6] The linux man pages project. *namespaces(7)*. Sept. 21, 2014. URL: http://man7.org/linux/man-pages/man7/namespaces.7.html (cit. on p. 8).

[7] Docker Inc. *What is Docker? An open platform for distributed apps*. 2015. URL: https://www.docker.com/whatisdocker/ (visited on 06/10/2015) (cit. on p. 8).

[8] Red Hat. *VENOM: QEMU vulnerability (CVE-2015-3456)*. Aug. 18, 2015. URL: https://access.redhat.com/articles/1444903 (visited on 06/21/2015) (cit. on p. 8).

[9] The linux man pages project. *capabilities(7)*. May 7, 2015. URL: http://man7.org/linux/man-pages/man7/capabilities.7.html (cit. on pp. 9, 14).

[10] The linux man pages project. *seccomp(2)*. Mar. 29, 2015. URL: http://man7.org/linux/man-pages/man2/seccomp.2.html (cit. on p. 10).

[11] GNU. *System Databases and Name Service Switch*. 2015. URL: `http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html#Name-Service-Switch` (visited on 06/18/2015) (cit. on p. 14).