UiO : Department of Informatics
University of Oslo

Observing impact of performance variation in cloud games using a chess engine

Jonas Sollihøgda

Master's Thesis Spring 2014

Observing impact of performance variation in cloud games using a chess engine

Jonas Sollihøgda

May 20, 2014

Abstract

Cloud computing is growing extensively, and in correlation, so is the number of users. Businesses look at the opportunity of increasing effectiveness and decrease cost, and are moving their infrastructure to the cloud. With such high increase in users, the cloud providers has turned to multitenancy. In which multiple tenants may end up running services or applications on the same physical server. This leads to shared resources, and may contribute to contention for resource allocation between the various services. This contention may result in varying degrees of performance and yield a very unpredictable service.

Furthermore one is witnessing parts of the industry taking advantage of the cloud as a platform for hosting games. The mentioned resource contention may impose severe performance deficiency on hosted games and servers running in the cloud.

This thesis propose the use of a chess engine as a way of simulating games hosted in a cloud environment where one is looking at observing the possible impact of shared resources and contention between virtual machines. The goal of the thesis is to map performance variation in the cloud and look at how it impacts the quality of the games, through observing chess matches being played under various conditions. In order to utilize a chess engine in the cloud, a set of frameworks was developed. The frameworks was responsible for hosting and running chess matches, and furthermore analyze the outcomes in order to observe any significant impact related to performance variation and resource contention.

Contents

List of Figures

Acknowledgments

I would like to express my sincere gratitude to my supervisors Paal Engelstad and Kyrre Begnum, for the dedication and guidance they have shown during this process. Your determination and encouragement has infused me with the inspiration and motivation needed in order for me to fully take use of my full potential. Your support has helped form this thesis, and for that I am eternally grateful.

Moreover i would like to express my sincere thanks and appreciation towards Hårek Haugerud. You have given me a tremendous amount of support throughout the course of this masters program, and your guidance and belief in me is highly valued.

I would like to thank my family for their support over the course of this thesis, your kind words and constant encouragement has been most helpful. And i am truly thankful.

Lastly, to the person who has stood by me through the entire course of this masters program, and constantly encouraged and pushed me in order for me to reach my goals, Karoline, i deeply thank you for all the patience and love you have shown.

Oslo, May 20th
Jonas Sollihøgda

Chapter 1

Introduction

Cloud-computing has grown extensively over the years and with it its number of users. Businesses and private actors look to the cloud in order to increase effectiveness and decrease cost. As a result cloud providers are faced with serving an increasing number of tenants and applications across their platforms, culminating in shared resources between the numerous tenants and hosted applications. Ultimately this lead to resource contention. This sharing of resources and resulting contention yields an unpredictable utilization of resources and potentially diminishes the cloud providers overall *quality of service*.

Still, adoption of cloud-computing is prominent. With cloud providers offering various cloud platforms such as Software as a Service(SaaS) and Infrastructure as a Service(IaaS), the incentives of moving to the cloud are many and the apparent benefits of increased agility, flexibility, scalability and the pay-as-you-go model are highly sought after by businesses and private actors. This has contributed to businesses moving away from traditional in-house server infrastructures and transitioned into moving more of the infrastructure and workload to cloud platforms.

It is not only within the corporate world that cloud-computing has become immensely popular, the gaming industry has also started to embrace the cloud platform as a way of providing their gaming services, so called "Gaming on demand". It is estimated that the online games market will double and that downloading/streaming games will increase a nine-fold by 2017[17]. Prominent providers such as OnLive, Gaikai and Ciinow provide end-users with an alternative way of playing games as the games are stored and run on the respective companies servers.

When one is concerned with game hosting in the cloud, awareness of CPU utilization is vital in order to deliver a solid product. In cloud environments where vast amount of the resources are virtualized and shared, the CPU of a physical server may be utilized by a various amount of hosted applications and services simultaneously, even more so in cloud environments which are run with multitenancy in mind. This may lead to contention, in practice meaning that VMs may struggle over the available resources. Therefore it is favorable to be aware of how the CPU is utilized across such a platform as it may help increase the predictability of how the CPU is used when being shared between multiple services and applications.

One may consider the scenario where one is playing a game, for instance a first person shooter(FPS), where the game itself is run within in a cloud environment. The game scenario features the player and one or more opponents in which are non-playable characters(NPC), who's actions are based on Artificial Intelligence(AI). Specific to this scenario is the expectation of the player towards the NPCs actions and level of difficulty. In this lies that the player who is facing the NPC expects the NPC to perform at the difficulty chosen. With this in mind, what may be the reason for the NPC in some instances playing at its intended level, and in other instances of the game underachieving and playing poorly when the player keeps repeating his pattern in terms of game play. For instance, one may be be playing a game at a specific difficulty and under optimal conditions one may be loosing 10 out of 10 matches, making the exact same choices every game. If one then was to repeat the game scenario at the same level of difficulty, the player would expect the NPC to win all of the matches again if the player makes the exact same choices as he did during the first set of matches. Although this time the player comes out victorious in 5 out of 10 matches, even though the matches have been played on the exact same terms.

Another game one may take into consideration is that of chess. When a player is playing chess versus the computer, one may make a move, and then observe the counter move made by the computer. Under the same conditions, meaning that the chess computer is running its evaluation at the same duration at all times, one would expect the chess computer to make the same move again if one was to experience the same game scenario in several matches. Although what if one in some cases witnesses the chess computer making a considerably inferior move to what one has observed earlier in the same scenario. The two situations described and their outcomes may be related to varying degrees of performance in terms of the underlying resources utilized by the NPC and the chess computer, and may be a result of possible contention in the environment in which they are run. The role of the system administrator in this case is making sure that resources always are readily available and keeping possible contention to a minimum in order not to experience such loss in quality in running games.

The predictability of performance regarding virtual machines(VMs) in a cloud is difficult to assess as several VMs running on the same physical server may share the same resources. As a cloud user one may not be aware of other simultaneously running VMs that are handled by other users on the same server. Furthermore it is hard to assess whether the resources of the underlying physical server are over utilized and as a result it is hard to predict the likelihood of VMs interfering with each other. Lastly, one is usually not aware of the activity of the surrounding VMs, making it difficult to predict if a VM will have enough resources available to accomplish its given tasks, without interference from its surroundings.

As the objective of this research is to map performance variation and look at how it may be linked to Quality of Service with respect to applications running in the cloud, with the main concern being CPU utilization between VMs, the usage of a CPU heavy technology is of interest. There is a strong correlation between the CPU and chess engines, as they depend on the CPU in order to evaluate chess positions. Therefore a chess engine is a viable option when wanting to have a CPU intensive application running in the cloud when mapping performance variation.

Chess engines have been around since the fifties, gaining more popularity through the seventies and eighties. As of today chess engines can be used by anyone who wants to analyze their chess matches and improve their game. A chess engine is first and foremost a computer program that analyses chess positions and decides what the best chess move is relative to the current board representation. Chess engines such as Houdini[20], Stockfish[32] and Komodo[5] are examples of highly popular chess engines within the chess community. A chess engine is a good candidate for the current research as they are well studied and fairly well documented, and they are furthermore simple compared to modern 3D shooters and other games one may have utilized for this purpose. Another important aspect is that it is possible to utilize the functionality of a chess engine through the development of scripts.

When it comes to using a chess engine within a cloud as a tool for mapping performance variation, one could look at the outcome of chess matches under various conditions, as a measure of how successful the engine has been in its position evaluations. One could coin this with a term such as *success rate*, in where one is able to look at the outcome of chess matches to make conclusions in terms of performance variation. With the underlying notion being if a chess engine is given less CPU its performance will be diminishing.

If one takes the above statement of predictability of performance in regards to VMs into account, two active VMs running on the same physical server may interfere with one another, although the process of this interference and how it manifests itself is uncertain. One aspect of this research is how the resources are utilized, more specifically the CPU. If the VMs are using CPU in some random intervals, they will most likely only interfere in the case where both VMs need to utilize CPU resources simultaneously. A different aspect may be where both of the VMs are constantly utilizing the CPU, in which case mutual interaction is constant, but it is not certain what effect this has on a running chess game. Would a 50% decrease in CPU utilization amount to a 50% decrease in success rate, or would a 10% decrease in CPU utilization lead to a 100% probability of losing a game?. None of the mentioned aspects are certain, and will as such be taken into consideration during the exploration in this research.

1.1 Problem Statement

The thesis and its involved research will be based on the following problem statement.

How can we explore Quality of Service(QoS) through the investigation of success rate of chess engines in a cloud environment with varying degrees of resources ?

In this research the aspect of *exploring Quality of Service(QoS)* refers to the observation and mapping of performance variation, more specifically in that of CPU utilization in a virtualized cloud environment. The performance of which will be observed is that of the services running on Virtual Machines hosted in a cloud environment, which comprises of looking at how various services on Virtual Machines performs when run in conditions where performance variation is introduced. The services running are not that of ordinary services such as web or mail, but services related to gaming.

Success rate in this research is not to be seen relative to the success rate of performance, stability or other system related values, but will be defined as a variable that describes or depicts whether a chess engine is playing at its full potential. The success rate will be used to determine whether or not the chess engine is playing favorably or whether it is playing in a more diminishing manner.

A *chess engine* in the case of this research should be thought of as Virtual Machine running one instance of a chess engine listening to a specific port. This will enable several VMs running such chess engines to partake in chess matches played across the network.

A *cloud environment* is in terms of this research an environment where physical resources are consolidated into virtual resources, shared between multiple applications and services being run on multiple Virtual Machines. The sharing of resources is done in a manner that is transparent to the hosted Virtual Machines.

Chapter 2

Background

Throughout the following background chapter general concepts of chess and its related jargon will be described in order for the reader to be comfortable with the various aspects of chess and how a game of chess is played. This will be necessary as the research conducted in this thesis will utilize a chess engine in order to answer the problem statement. It will furthermore introduce some concepts in relation to cloud computing and virtualization since it is imperative knowledge in order to understand the architectural design of the current research.

2.1 Performance Variation and unpredictability

As the demand for serving an increasing number of users has grown, cloud providers have adopted a multitenancy architecture allowing multiple tenants and applications to use the same physical and network infrastructure. Some virtualization techniques do not provide effective performance isolation between virtual machines [26], hence when multiple tenants and applications compete for resources of a physical server, it introduces the problem of resource contention. Such a contention implies that VMs will be competing for resources such as CPU, memory and disk. With VMs competing for resources another issue arises, that of unpredictability in terms of how much resources are in fact being used by the respective VMs at any given time.

With the cloud market being what it is today, performance variation is present within many cloud environments as many applications may be running simultaneously within vicinity of one another on the same physical host. It has been proposed that cloud providers re-evaluate their current Service Level Agreement policy as it is mainly concerned with uptime and availability, and not the underlying performance in which the consumers trust is provided[10]. Studies have been made in order to map performance variation within various Cloud environments such as Amazon EC2[16] and Googel App Engine[18][23].

2.1.1 Virtualization

Virtualization is a concept that started with IBMs mainframe dating back to the late 1960s and early 1970s. It was in this period of time where IBM was looking into the possibility of sharing the mainframes computer resources among several users. This technology has since been popularized and is widely used within the field of cloud computing. The goal of this effort was to improve the overall efficiency of both the users and computer resources. As of today large data centers embrace this virtualization technology as it enables them to make abstractions of the underlying physical hardware and create larger aggregated pools of virtual resources. These resources may be CPU, memory and storage[21].

The virtualization itself is achieved by installing a *virtual machine monitor(VMM)* also knows as a *hypervisor* on a host machine. The virtual machines which are running on top of the hypervisor are regarded as *guest machines*, and as a way of managing the execution of these guest machines the hypervisor provides them with a virtual operating platform. The created guest machines will now have the ability to share the virtual hardware resources. There exist different types of virtualization techniques and ways the hypervisor is employed, one of which is *full virtualization*, meaning that the hypervisor is installed directly on top of the underlying hardware making it possible to manage the hardware and guest machines. The guest machines will run on top of the hypervisor. Another virtualization technique is that of *paravirtualization*, where the hypervisor is run within another traditional operating system, and guest machines are run on the layer above the hypervisor. The virtualization techniques can be seen in Figure 2.1.

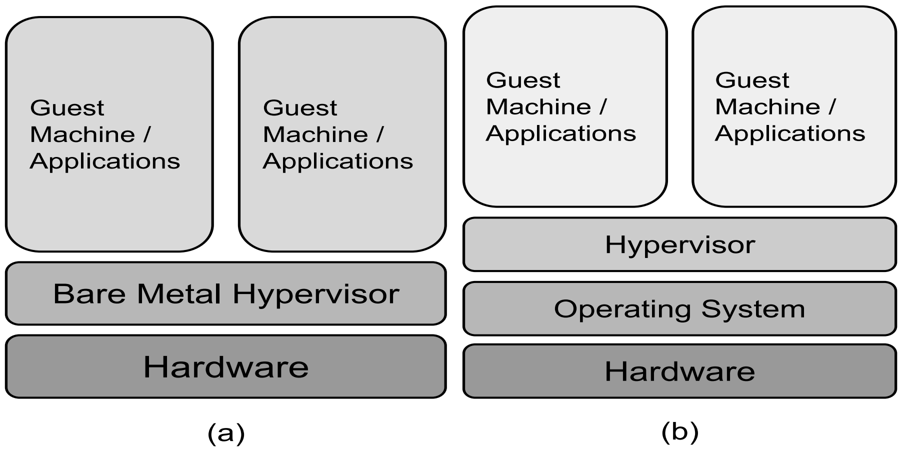

Guest Machine / Applications
Guest Machine / Applications
Bare Metal Hypervisor
Hardware
(a)
Guest Machine / Applications
Guest Machine / Applications
Hypervisor
Operating System
Hardware
(b)

Figure 2.1: Illustratring Type 1, Full Virtualization (a) and Type 2, Paravirtualization (b).

2.1.2 Cloud-Computing

Often associated with virtualization is *cloud computing environments*. Cloud computing is a phrase that was coined for a concept involving application and service hosting across the internet and furthermore the underlying architecture on which it is offered upon. Amazon and Rackspace are examples of providers who offers both public and private clouds for consumers. Consumers have the opportunity to choose from different service models:

Software as a Service(SaaS) is a service model where consumers are provided with applications fitting their requirements by cloud providers. The cloud providers handle the infrastructure and the overall platform that run the application software and the consumers only need to connect to the application software through thin clients or other cloud clients.

Platform as a Service(PaaS) gives consumers the opportunity to move, develop and run their own software on their own platform within the cloud environment.

Infrastructure as a Service(IaaS) is a service model that enables the consumer to create and run virtual machines within a cloud environment and furthermore take control of resources such as storage and network. Additionally the consumer is given access to resources such as firewall, load balancing and software bundles. Cloud providers are able to offer this on a "on-demand" basis, meaning that consumers have the ability to scale their resources in terms of their requirements. The VMs that are created within the cloud are under the control of consumers themselves, and they can administer them as they would any pay-as-you-go machine.

As of recent years some of the focus within the area of cloud computing has shifted towards that of gaming in the cloud. Unlike the other "as-a-service" models such as Software as a Service and Infrastructure as a Service, a new model has been proposed and is called *Gaming as a Service(GaaS)*. Although GaaS is considered new, it has been around for some time as games have been offered in the cloud for a longer period through sites such as Facebook, offering games from companies such as Zynga[33]. Zynga released Farmville on facebook and garnered huge success within a short time span[34]. Furthermore companies such as OnLive offer consumers games that are playable in the cloud on both computers, consoles and other devices.

2.2 Environment of implementation

The environment in which will be utilized during this research is that of an OpenStack[19] cloud environment. The OpenStack environment provides the the user with the following possibilities:

- Create Images
- Choose capacity of the virtual machine(Flavors)
- Creation of snapshots

In the OpenStack environment, when creating a virtual machine(VM), one has the possibility of choosing from several images, meaning one may choose various operating systems that the VMs will be run with. The following research will be using a Linux/Unix based image. Moreover one has the possibility of choosing from predefined capacities in which these VMs will be fitted with. The flavors dictate the storage, memory and CPU capacity of a given VM. Furthermore as the VMs are being created with different specifications, for different purposes, it is possible to create snapshots of the already created VMs. These snapshots are in essence a duplicate of the original, creating another instance with the exact same specifications in terms of capacity and data contained within the original VM.

2.3 The game of chess

Chess is a strategy based board-game in which two players alternate moving a set of chess pieces with the aim of capturing the opponents pieces, and ultimately check-mating the opponents king. The traditional game of chess is played on an eight by eight chess board with each side, black and white respectively, having 16 pieces to choose from. Each game of chess is begun with the two players having one *king*, one *queen*, two *rooks*, two *knights*, two *bishops*, and eight *pawns*. All of which have different movement patterns. The overall goal of a traditional chess game is to checkmate the king of ones opponent, which means forcing the king into a position where it may no longer move without being captured [6].

A game of chess is divided into three phases, the opening, middle game and end game respectively. These phases have different characteristics.

The opening phase is the initial start of a chess game, in which a set of initial moves in a recognizable sequence are made. Although *opening* is used as a general term within chess, an initial move for white is considered *openings* and *defenses* for black. There exists many variations of chess openings, and many of which are named and more popular than others.

The middlegame is a phase recognizable by the many different board combinations and tactical moves made by each side. There is no distinct beginning or end of the middlegame. The opening moves may be part of this phase, and the middle game will after a while transition into what is called the end game.

The end game is generally categorized as the phase where the amount of captured chess pieces grows, leaving few pieces left on the board. The end game will conclude when one of the players check mates the opponents king, or the two parties agree on a remis, which is a draw in a game of chess.

2.3.1 Algebraic Notation

Standard Algebraic Notation(SAN) is the only system of notation recognized by the World Chess Foundation, and is also considered the standard way of recording moves in a game of chess. The rows(from left to right) on a chess board is commonly referred to as *files*, given the letters a through h respectively and the columns are referred to as *ranks*, represented by the numbers 1 through 8. Bottom to top for white, and from top to bottom for black. Such that a specific position on a chess board is denoted with both a file letter and a rank number. E.g *a2,b3,c7,d8*. Figure 2.2 illustrates this.

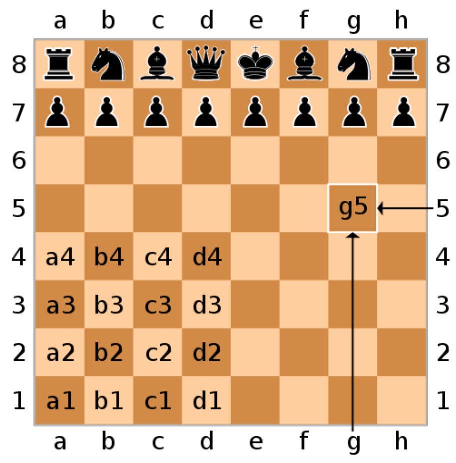


Figure 2.2: Illustratring Algabraic notation[7][8][9]

The Algebraic System denotes the six different chess pieces by their first letter;

- K = King
- Q = Queen
- R = Rook
- B = Bishop
- N = Knight(N used not to conflict with kings K)

The pawn piece is not denoted with any letter, and a pawn move will be recognized only by the square to which it is moved. Each move made during a game of chess is represented by the first letter of the name of the piece and by the square it is placed. E.g: Knight to c6 = *Nc6* and King to a2 = *Ka2*. As mentioned a pawn move is only indicated with the square of which it is placed, for instance moving a pawn piece to e4 is only denoted *e4*[22].

2.3.2 Board representation

In chess there is a standard for noting the current board position for a game of chess. *Forsyth-Edwards Notation(FEN)* is a system invented by David Forsyth and is considered the standard for noting board positions in chess. The FEN notation or record, holds information about the various piece placements currently applying to a specific board, it furthermore holds information regarding which player is next to move and how many moves are made. A FEN is also used to restart or initiate a game from a specific game position. A typical FEN record will look like this:

The FEN record, represented in ASCII, consists of 6 fields separated by a space, each field representing various information about the game. The fields are as follows:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
                       ¦                    ¦   ¦   ¦ ¦ ¦
                       1                    2   3   4 5 6
```

Figure 2.3: The concept of a FEN string

1. Placement of the pieces. All of the ranks are described, 8 through 1. Each *rank* is described from *file* a through h. Each piece is noted using the Standard Algebraic Notation. White pieces are denoted using upper-case letters(PNBRQK) and similarly black pieces are denoted using lower-case letters(pnbrqk). Each empty square within a rank is denoted using digits 1 through 8, and / separates the ranks.
2. Tells which player is next to move; w for white and b for black.
3. Denotes castling availability. If either white or black can castle, the field is represented by one or more letters: "K" for king side castling or "Q" for queen side castling for white. As for black the same will be denoted, only in lower case letters. In the event where castling is not available the field will simply denote "-".
4. Represents the target square for which *"En passant"* can be made. If there is a possibility for en passant, the field will be denoted with the target square in Standard Algebraic Notation E.g *b3*. If no such move is available it will be denoted with "-".
5. Halfmove clock. This field represents the number of halfmoves since a pawn advance or last piece capture. The field is used to check if it is possible to call a draw under the fifty-move rule.
6. Fullmove number. This is the number of full moves that has been made, it starts at 1 and is incremented with each black move.

2.3.3 Evaluation function

The evaluation function[13] within a chess engine is used to heuristically determine the relative value of a position in chess. This value represents which of the playing parties are currently at an advantage after a move has been made during a game of chess. The evaluation function bases its evaluations(which are approximations) on the balance of *material* for both sides in conjunction with other considerations, with material being the most significant aspect.

Material is a term that is determined by the sum of piece values for both white and black, in which is the sum of constant values for each piece still on the board, measured in units of a fraction of a pawn. This is commonly referred to as the *centipawn scale*[15][14][12]. Commonly the *chess piece relative values* for chess pieces are[30];

- Pawn = 1

- Knight = 3
- Bishop = 3
- Rook = 5
- Queen = 9

Centipawns(cp), then gives a score in which 100 cp = 1 pawn. Although in regular chess the value is of no specific help, but may be useful to the players. As mentioned the centi pawn value gives an indication as to which player has an advantage on the current board, it may also happen to be an equal standing in which none of the players have any form of advantage. As a game of chess progresses this centi pawn value will shift in favor of the player who has the stronger position on the board, even more so if one party collects material from its opponent. One may look at the standing at the start of a match, in which the centi pawn value would be 0, indicating equal advantage at the beginning of the match(Although white is considered to be at an advantage as he has the first move). As the players move their respective pieces on the board, the centi pawn value will increase or decrease based on how good the previous move was. During a chess game, if the centi pawn value is above 0 then white has the advantage, accordingly if the centi pawn value goes below 0 and becomes negative, the black player is currently holding an advantage.

2.3.4 Stockfish Chess Engine

The Stockfish chess engine is derived from Glaurung[28], an open source chess engine developed by Tord Romstad, and is a collaborated work between chess programmers Tord Romstad, Marco Costalba and Joonas Kiiski. Stockfish is, as of writing this thesis, the second strongest chess engine in the world, only beaten by the proprietary chess engine Houdini[3][1].

During the thesis Stockfish will be run in a Unix-like environment and it is possible to compile it to suit the environment in which it is selected to run, which is recommended as it improves the efficiency of the chess engine. As of now Stockfish supports the use of 64 CPUs running on either a 32 bit or 64 bit architecture, with the possibility of changing the wanted number of CPU cores when running the engine.

2.3.5 UCI Protocol

Stockfish is what is commonly referred to as a *UCI chess engine*. The *Universal Chess Interface(UCI)* is a protocol used for the communication between a chess engine and a chess user interface[24][25]. Stockfish supports the UCI protocol, and traditionally one would communicate with the engine through a chess GUI such as Arena[11]. All the communication between the engine and the GUI is done via text based commands.

As the communication is done via commands one has the possibility to communicate directly with the UCI protocol through a regular command line interface. The usage of the command line interface will be used in the case of this thesis, as UCI commands will be passed in order to tweak engine parameters and invoke various functions within the Stockfish engine.

2.4 Related research

Cloud computing has seen a great increase in popularity, and has as such also become the focus of many researchers. Within the area of performance variation and performance predictability in the cloud, similar research to that of this has been conducted. Research of which considers performance variation and predictability to that of CPU utilization in the cloud, is considered that of similar research.

When it comes to that of mapping performance variation in the cloud, a proposed research is Schad et al.[29] in which CPU performance is measured using Unix Benchmark Utility, which is a widely used tool for measuring CPU performance. The Unix Benchmark Utility provides a single CPU performance score by making various concurrent integer and floating point calculations. The measurements of the proposed researched where made in the Amazon EC2 cloud in order to map the variance of performance and analyze the impact it had on real applications. Furthermore more, Moses et al.[27] propose the use of shared resource monitoring in order to understand the resource usage of each virtual machine on a given platform. And they conclude that high priority applications suffer from contention if scheduling of resources is not done at a data center level.

Other research such as Silva et al.[31] have delved into the area of enabling performance isolation as means of controlling how multiple VMs share resources on a physical host, in order to be able to classify different levels of QoS to the customer. They look at how they can employ "soft-limits and "hard-limits" enforced by the underlying hypervisor such that it is possible to set a lower and upper bound of resource usage. Their preliminary experiments found that the developed framework effectively enforced CPU and Network I/O limits and protecting performance of a virtual machines workload while maintaining a high resource utilization towards the numerous hosts.

Chapter 3

Methodology

The methodology chapter will introduce the *approach* of this research, which will detail how the following problem statement will be answered:

How can we explore Quality of Service(QoS) through the investigation of success rate of chess engines in a cloud environment with varying degrees of resources

In the problem statement several aspects are of significant interest, and may be further denoted into key concepts: *Measuring success in the context of QoS(K1)*, how may one define and view success in conjunction with resource contention and how does it affect QoS. *Incorporating a chess and analysis framework in the cloud(K2)*, the creation of a chess framework is needed in order to utilize a chess engine for the purpose of this thesis. And the subsequent analysis framework in order to analyze the data produced by games of chess.

The focus of the design will be on answering, or accommodating the concepts mentioned in this brief introduction and furthermore serve as the overall scheme for the development of the frameworks that will serve to answer the concept of K2.

3.1 The objective and design

The goal of this thesis and research is to design and create a framework for observing and mapping the impact performance variation has on quality of service(QoS) of applications running on virtual machines based in a cloud environment. As described during the introduction of this thesis, a subsequent goal is being able to make a cohesion between resource contention occurring in such environments to that of success in a game. Success in the case of this research, related to gaming, is being measured in how well a specific chess match is played out by a chess engine.

The framework in question will have to be able to integrate a chess engine within a virtualized cloud environment, and furthermore be able to orchestrate chess matches on a large scale across the network. It will be necessary to identify the technical possibilities of the chess engine as the framework will have to rely heavily on automation, since manually arranging a large scale of chess matches and storing the related data is a non-trivial task. Accordingly the framework will have to take into account the preservation of required data from intended chess matches to be played. The data will need to be stored in such a fashion that one may easily extract only wanted results for analysis.

Furthermore, the method of storage to be used within the architecture must feature a way of backing up or replicating the created storage structure and its contained data.This may be done manually. The framework that will be proposed also needs to account for an analysis module that will be utilized by possible third party nodes, proposed as *referees*, as a part of observing and comparing the impact that may have been imposed by varying degrees of performance.

3.2 Design Phase

In order to comprehend the framework and its related architectural design, a way of portraying the said design is needed. Models may provide a good overall view of the frameworks intended functionality and its required modules. The complexity may be viewed in a more shallow manner and ease the comprehension of the framework.

3.2.1 Modeling

Whether you are developing a tool, creating a framework, or building an entire architecture, a model or graphical representation of the intended creation is always preferable as it may give both the developer and exterior onlookers a better perception of the end product. A model may give easier insight into the how the processes within a given framework intertwine, and how they are needed to give life to the intended functionality. It will furthermore help introduce the various tasks that the intended functionality are meant to serve. Although a modeled design may be helpful in most cases, it is important to recognize the possible flaws it may possess. One has to be careful not to make the models to extensive as this will ruin the purpose of the models intention of reducing complexity and ease the overall comprehension of the processes, modules or in worst case the entire framework.

In order to make a model or graphical interpretation of a proposed framework one first has to decide on what type of modeling language one wants to make use of. This is important as it will determine how one structure and build ones models. It will furthermore set limits for what is possible to achieve given the constraints of the chosen modeling language.

When using models or graphical representations of a proposed module or feature, people that are either exterior to the project or just not fluent in any sort of programming languages may have a chance of comprehending what the displayed model is trying to achieve in terms of either how a specific process is thought out to work, or how a certain module will be developed. An example of said modeling languages is UML(Unified modeling language). UML may be used to model a variety of concepts of which some being architectures, processes, as well as data structures. A model may manifest itself in different forms of diagrams such as state diagrams which serve to describe and display a behavior in a given system, flowcharts which easily depicts the flow of a given process within a system or a sequence diagram that visually illustrates how certain processes act and operate with each other. UML will be used throughout the research to model the architectural concepts and furthermore to model certain proposed functions that will be implemented within the frameworks.

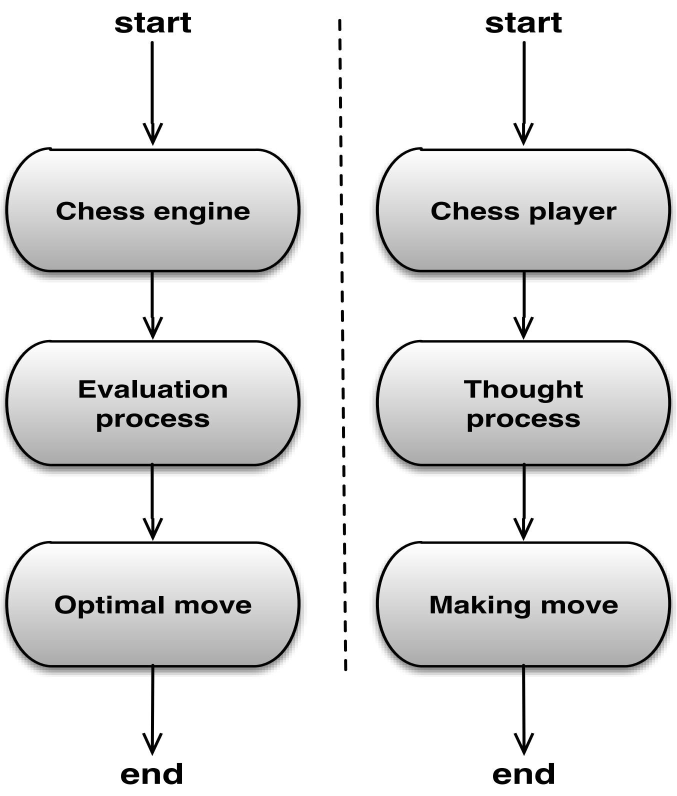


Figure 3.1: Illustrating the perception of a chess game through a chess engine and a human player. The design formalized using a UML model.

3.2.2 Pseudocode

Another approach to that of illustrating a concept or some piece of functionality is that of pseudocode. Pseudocode may be seen as a high level description of a given process or feature(or functionality). Pseudocode, in its simplicity uses the same structural conventions to that of a programming language. The main difference being that pseudocode is intended for human reading, as opposed to that of a programming languages which is to be understood by a machine. With that being said, pseudocode may be interpreted easier by a person that has some experience with programming contrary to that of a person who has no or limited exposure to programming languages.

Models or diagrams may be suitable for many occasions, but it may be difficult to really grasp the finer details of a specific process or function. If one combines a model with a piece of pseudocode one may be able to illustrate the overall meaning of a given process, but in order to understand how the process works on a deeper level one may introduce pseudocode that explicitly details the process in a more specific manner. That way one may not only understand the overall design, but also comprehend how the following design may be implemented.

```
WHILE match is going
  AND IF new match
   THEN
     INSERT information to database
     PRINT inserted information
  END
CONTINUE match
```

The pseudocode displayed above tries to explain a certain function within a program. In comparison to a model, these few lines of pseudocode takes up less space and may even more accurately portray what at a given function is supposed to accomplish. Although it is possible to combine the two in order to have an overview of the function and its integration with other parts of the framework and supply it with a snippet of code to further detail the functions purpose. Pseudocode will be used as it provides an easier way of portraying and explaining various functionality on a lower level, and together with a model it serves as a more complete look on the wanted, or implemented piece of functionality.

3.3 Implementation Phase

The modeled design will serve as the overall scheme for the process of implementation. The implementation will see the creation of a prototype that will provide answers to that of the concepts in K1 and K2.

3.3.1 Environment

As for the implementation, the framework(K2) will be based on the previously mentioned design and as the current research involves observing and mapping how performance variation may impact quality of service, and subsequently this could also be linked to how to better ascertain the predictability of the extent to which one can expect that performance varies.

If time permits, the most ideal way of answering the problem statement will be to integrate the proposed framework within a cloud environment, as one wants to recreate a scenario that is as close to realistic as possible. Usually one may want to avoid placing the proposed solution in a real environment as it may expose the solution to unwanted noise which may manipulate the results and somewhat distort the process, but in the case of the frameworks developed here, this noise will be welcomed as it servers as the main source of distortion in order to produce the wanted results for answering the questions in regards to performance variation and the unpredictability that it brings along.

Although a realistic environment is ideal, there are some aspects that need to be considered before one can start doing live testing of the framework, as it is uncertain how the quality of service will be affected by the possible performance variation. Furthermore there is an underlying uncertainty as to what one may expect to observe during testing. As a result a form of rigged testing is needed before moving the framework into a real environment and commencing live testing. This rigged, or synthetic testing will serve as blueprint for the live testing of the proposed frameworks.

3.3.2 Synthetic testing

Before one may one commence any sort of live testing in a more real like environment, there are some preliminary tests that need to be conducted. The reason, as already mentioned, is that it is uncertain how the variation in performance will impact, as in the case of this research, a game of chess.

Furthermore, in order to map the conjunction between the key properties of K1, and how one expects to observe their coherence and behavior, a few games will be played in a more controlled environment and where the games themselves are controlled to a greater degree in order to see if the observations made of the properties are like the ones that was expected. It would have been impossible to commence with the live testing immediately, but it would have be difficult as there are uncertainties tied to the observations of a specific chess game and its related properties.

The synthetic testing will be organized in a more controlled fashion, meaning that instead of having real-time noise interfering with the undergoing chess matches, a given penalty will be introduced as a way of mimicking unwanted noise. This may be viewed as a way of controlling the CPU usage from inside the virtual machine where the chess engine is running, as a penalty inflicted on its evaluation time will result in decreased time for the chess engine to evaluate the given position, contrary to the time it was originally meant to spend evaluating. It is fair to argue that inflicting such a penalty would be close to what one would expect to happen in a real environment as resources may be shared and applications may be experiencing contention, and as a result consume less CPU than expected.

3.4 Approach summary

This chapter has established an approach for the research that is to follow. The approach serves to give insight into how one will go about answering the problem statement, and how one is to obtain a cohesion between the success of games to that of contention in the cloud, and furthermore bring an understanding as to how the proposed frameworks will be modeled and lastly where and how they will be implemented and created. The following research will be comprised of the following phases:

3.4.1 Exploration

Identifying technical phases

As mentioned during the approach it is crucial to identify the technical possibilities of Stockfish in order to be able to create the desired frameworks, and because the frameworks will rely on automation. In terms Stockfish, this may be how to successfully establish communication, setting a specific FEN, initiating the evaluation and invoking moves.

Modeling phase - Chess framework

The main objective of the modeling phase is to identify how one may orchestrate chess matches on a large scale. The modeling phase will propose a solution as to how one should create and implement the intended chess framework. This phase will comprise of identifying a set of criteria that needs to be met during the development of the chess framework, but it will also seek to identify the various functionality needed in order for the framework to successfully work. A need to identify how one can orchestrate chess matches is in order, furthermore how one is to control the framework, identifying the data provided by Stockfish and how one is to store these data.

Implementation phase - Chess framework

As for the implementation phase, the objective will be to create the proposed frameworks based on the criteria set in the modeling chapter and furthermore elaborate on how the various concepts of functionality are sown together. Important factors what will be considered are the creation of chess servers, how to implement the needed functionality and looking at how a chess match may be performed by the developed framework.

Modeling phase - Analysis framework

As mentioned during the approach(K2), two proposed frameworks will be developed. The modeling phase for the analysis framework will set of criteria for how to successfully analyze the data produced from the chess matches. This may be how to organize the analysis, running and controlling the proposed analysis framework, how to fetch and store the analyzed data.

Implementation phase - Analysis framework

The purpose of the implementation phase concerning the analysis framework is to create and implement the analysis framework based on the criteria set during the modeling of the framework, and furthermore look at how an analysis is conducted.

3.4.2 Investigation

Testing

As set of various synthetic tests will be conducted in order to observe and map the variation in performance and conjunction between the key properties of K1. The synthetic testing will serve as a blueprint for live tests, that will only be conducted if time permits.

Analysis

The analysis phase will observe the generated data from the various chess matches as well as the analyzed data. The purpose of the analysis is to look at the properties in K1 and link them to the results received from the rigorous synthetic testing. The analysis will look to answer if one is observing the wanted conjunction between that of success in games and resource contention, in this case in terms of the synthetic test. And if there are noticeable variations in the quality of the moves made by the players?

3.5 Expected Results

The main tasks are to successfully identify variables capable of being used as criteria of success in order to map the cohesion between a success in games to that of resource contention. Furthermore the identification of technical possibilities in terms of the Stockfish chess engine, as it will be vital for the successful development of the proposed frameworks. The research will be formed around several phases of which will lay the foundation and ultimately shape the resulting frameworks. One of these phases is a modeling phase which will identify the needed functionality, based on the technical possibilities, and where the aim is to model a working architecture in order to pursue the goal of creating a fully functional chess and analysis framework.

The following process of creating the frameworks will need to adhere to certain criteria established during the modeling phase in order to be successfully implemented and working. The synthetic testing will serve to identify and fulfill the expectations surrounding the variables concerned with uncovering the cohesion between success and resource contention, and will furthermore serve as a blueprint for the live testing, as will only be conducted if time permits.

Chapter 4

Result - Identifying technical possibilities

This chapter seeks to identify and map the technical possibilities of the Stockfish chess engine. This process is necessary in order identify what is possible to achieve with a chess engine when one thinks of utilizing it in a virtualized cloud environment, and when wanting to orchestrate chess matches on a large scale across the network. It is furthermore important to ascertain the technical boundaries of Stockfish as the proposed frameworks will rely heavily on automation, and therefore it is vital to know if there are any limitations to Stockfish that may hinder or complicate the matter of automation.

4.1 Introduction of terminology

During the remainder of this thesis, specific phrases related to the world of chess, chess engines and its accompanying features will be frequently used. As such a brief introduction of its jargon is in order to fully comprehend the technology and research conducted in the following chapters.

- *Stockfish* is the chess engine used during this research, a chess engine may be viewed as a tool for evaluating chess games and its related moves.
- *UCI protocol* is the main protocol used for communicating with a modern chess engine.
- *Ply* or half move, refers to that of a move made by ONE side only(E.g whites move). Consider the following; Stockfish performs a 6 ply search, in reality it has searched 3 whole moves, like 1. e2e4 e7e5, 2. g1f3 b8c6, 3. b1c3 g8f6.
- *Black & White* refers to that of the participating players of a chess game.
- *FEN* is a string that represents a specific chess board with a specific set chess piece positions.
- *Evaluation function* is Stockfish's way of analyzing chess piece positions.

- Best move is the last thing Stockfish outputs when its evaluation is over, indicating what it considers to be the most optimal move to make in the current game situation, hence the best move.
- *Stdout* or standard out, refers to that of a stream of data returning from a running process, usually returned to the screen of a user.

4.2 Selecting a chess engine

A chess engine is in essence a piece of computer software that analyze chess positions and ultimately decide what the best move would be for the moving party. Chess engines have increased in popularity and own tournaments, such as the *Chess Engines Grand Tournament*[2], are being held in order to rate and decide which one of the engines are the strongest.

As of today a number of chess engines are available for use, which are continuously being updated and improved. Houdini[20], Stockfish[32] and Komodo[5] are examples of chess engines that are currently considered the strongest chess engines on the market[3][1]. For this thesis Stockfish has been chosen as the engine to use as it is an open source project licensed under the GNU General Public License[4] and can be run in a unix/linux environment.

4.3 Communicating with Stockfish

In order to have Stockfish perform various actions like running an evaluation or making a move on behalf of a specific player, one is in need of a way to communicate with the engine. As mentioned during the background chapter of this thesis, chess engines are usually referred to as *UCI chess engines*. What lies in the term UCI chess engine, is that it is possible for the engine to communicate with a chess graphical user interface also known as a GUI, through text based commands that are passed to the engine when the user interacts with the GUI through the UCI protocol. The GUI on the other hand is not necessary. If one compiles Stockfish on a given machine one has the possibility to run the engine and communicate with it directly through feeding it commands via a terminal, and it is possible to read the output of the commands through *stdout*.

As can be seen in figure 4.1 it is possible to issue a command using the UCI protocol that will invoke a specific function within Stockfish, be it running an evaluation of the current position or making a move.

Stockfish is the most integral part of the proposed framework and as such it has been necessary to identify how one may be able to interact with the chess engine to receive the desired behavior and functionality. Important factors in this process are identifying how one can have Stockfish set a specific FEN, or more simply put, the current board representation, and consequently how one may analyze the given FEN. Furthermore one has to, based on the evaluation given by Stockfish on the

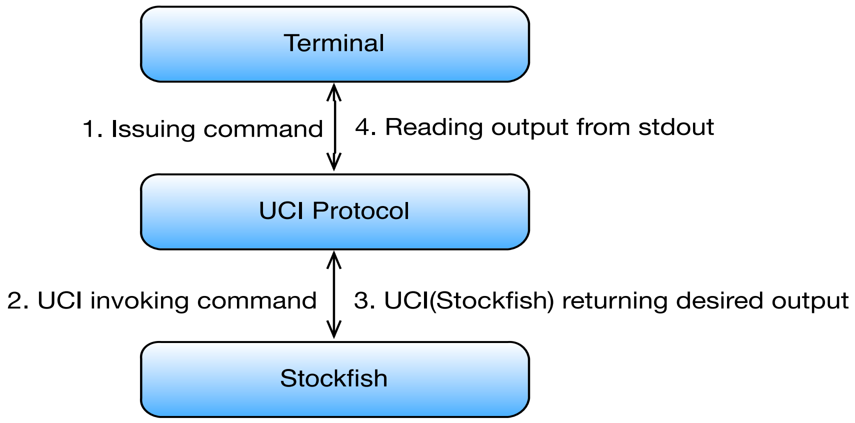

Terminal
1. Issuing command
4. Reading output from stdout
UCI Protocol
2. UCI invoking command
3. UCI(Stockfish) returning desired output
Stockfish

Figure 4.1: The above figure illustrates the concept of communicating with Stockfish through the UCI protcol.

said FEN, be able to move the piece of which was found to be the best move for the currently playing party.

4.3.1 Setting a specific FEN

An important step in creating the needed framework using Stockfish is knowing how one may analyze a given position on the chessboard, and using the UCI protocol it is possible to either feed Stockfish with a desired FEN string, or one may just start the engine and it will automatically assume a new game is about to begin, meaning that it will set a board in which no pieces have yet moved. In order to instruct Stockfish as for what FEN it is to set, one has to issue a UCI command that specifically invokes a new FEN string within the engine. One may see the needed command below:

Setting a FEN
```
position fen rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

If one inputs the above command using the UCI protocol, the FEN string within that command will be set and by issuing a debug command, *"d(debug)"*, one has the opportunity to both display the current board representation in cohesion with the set FEN and obtain information regarding legal moves that are possible to make and whether or not there is a check mate in a given number of moves.

Figure 4.2 displays how Stockfish would portray the currently set FEN when running the debug command.

4.3.2 Initiating the evaluation

After one has set the wanted FEN, it is possible to invoke an evaluation of the current board. The evaluation done by Stockfish may either be done in a given time frame e.g 5 seconds, or one may give Stockfish a specific depth in which to analyze. In order to invoke the evaluations one has to pass the following command to Stockfish:

```
+---+---+---+---+---+---+---+---+
| r | n | b | q | k | b | n | r |
+---+---+---+---+---+---+---+---+
| p | p | p | p | p | p | p | p |
+---+---+---+---+---+---+---+---+
|   | . |   | . |   | . |   | . |
+---+---+---+---+---+---+---+---+
| . |   | . |   | . |   | . |   |
+---+---+---+---+---+---+---+---+
|   | . |   | . |   | . |   | . |
+---+---+---+---+---+---+---+---+
| . |   | . |   | . |   | . |   |
+---+---+---+---+---+---+---+---+
| P | P | P | P | P | P | P | P |
+---+---+---+---+---+---+---+---+
| R | N | B | Q | K | B | N | R |
+---+---+---+---+---+---+---+---+
```

Figure 4.2: The above figure illustrates how Stockfish represents a chessboard after a FEN has been set.

```
Initiating an evaluation by either time or depth
go movetime 5000(ms)

go depth 10(Ply depth)
```

The evaluation Stockfish performs is done in such a way that if one specifies a specific depth, e.g 10 like displayed above, Stockfish will search 10 plies(half moves) ahead and while this evaluation is running Stockfish will output the specific result for each depth it has evaluated. At the end of every depth evaluated, Stockfish will return what it has found to be the best move to make based on the evaluated position. E.g an evaluation of position e2e4, may result in Stockfish recommending moving e2e5(Which is moving a pawn forward one square). Similarly if one desires only to evaluate within a certain time frame one may invoke the *go movetime* command which tells Stockfish that it shall only evaluate for the duration of the specified time. The time specified is in milliseconds and as such if one evaluates for 5000 milliseconds, Stockfish will run its evaluation for 5 seconds. Running an evaluation within a specific time may result in different search depths as Stockfish will stop its evaluation when reaching the specified time.

Figure 4.3 shows an example of the output from an evaluation conducted by Stockfish.

```
info depth 1 seldepth 1 score cp 75 nodes 27 nps 13500 time 2 multipv 1 pv e2e4
info depth 2 seldepth 2 score cp 12 nodes 140 nps 46666 time 3 multipv 1 pv e2e4 e7e5
info depth 3 seldepth 3 score cp 57 nodes 459 nps 91800 time 5 multipv 1 pv g1f3 d7d5 d2d4
info depth 4 seldepth 4 score cp 17 nodes 977 nps 122125 time 8 multipv 1 pv e2e4 d7d5 b1c3 g8f6
info depth 5 seldepth 5 score cp 50 nodes 1842 nps 153500 time 12 multipv 1 pv d2d4 d7d5 g1f3 g8f6 b1c3
info nodes 1842 time 12
bestmove d2d4 ponder d7d5
```

Figure 4.3: Example of output from an evaluation of Stockfish

4.3.3 Invoking a move

As Stockfish has finished evaluating the given depth or specified time, it will return a subsequent best move in which one wants to invoke on the current board. In order to invoke this move, one may use the following command:

Invoking a chess move
```
position fen rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1 moves e2e5
```

As displayed above one may take note that in order to make a specific move, the UCI protocol demands that the current FEN is a part of the input in order to make sure that the move is made on the correct board, and that it is a legal move being made on the represented FEN. Making a move that is not legal will result in Stockfish returning an error. Stockfish separates the actual evaluation of moves from that of invoking the concurrent best move, as Stockfish will run its evaluation and output the desired best move and will then sit idle until the move is invoked by the command displayed above.

4.4 Determinism

Before the synthetic testing commence there is a need to establish whether or not the the Stockfish chess engine behave in deterministic manner. One wants to confirm that there is no random events being triggered during run time. This means that if Stockfish is set to evaluate a match with specific settings and running at a specific depth, one should obtain the exact same result if the match is repeated. Meaning that Stockfish will evaluate in the same way each time, ending up with the same best move, traversing the same number of nodes and so forth for all of the values related to that specific evaluation. This is important knowledge as it will be crucial to know how the chess engine behaves under certain conditions in order for the testing to be valid. One will have to be certain that the specific tests that are run, is running on the same terms, as far as the chess engine is concerned. The outcomes of the chess matches should not be determined because of the chess engine, but on the noise factor imposed on the participating players of the chess match.

Chapter 5

Result - Chess framework

5.1 Modeling and architecture

The following chapter will introduce how the designed chess framework and architecture was created and deployed.

In an effort to answer the problem statement given in the introduction of this thesis, the following chapter will detail the process of modeling and designing the needed chess framework and that of the infrastructure it will be run within.

5.2 Orchestrating chess matches on a large scale

When planning to orchestrate chess matches on a large scale as envisioned in this research, there are certain criteria that need to be met during the development of a framework. The criteria are as follows:

- A way of organizing a large array of matches with different parameters.
- The creation of a chess bot.
- A module responsible for handling communication between chess bots.
- Looking at the possibility of parallelization of matches.
- Identifying data provided by Stockfish and what is of importance.
- A means of storing data produced by Stockfish.

5.2.1 Organizing matches in batches

As the preliminary chess matches run during the synthetic testing will have noise introduced in varying degrees, one will be faced with matches inhabiting different characteristics in terms of evaluation time, and one may witness varying results based on what type of penalty is given to a specific match setup. Furthermore if time permits the proposed framework will at some point the placed in a live environment, where the matches will be subject to various degrees of noise, and there will be scenarios where the chess servers are running at different capacities.

As a consequence, one will have a diverse group of matches being played, and thus it is imperative that there is a way of organizing the various matches in such a way that one is in control of what sort of parameters the matches have been initiated with. A way one may do this is creating unique batches in which to store specific matches. One may think of a batch as putting each individual and identical initiated match into the same set, which will be identified by a set of unique identifiers along with its given parameters(Such as penalties, evaluation time).

By organizing the matches in such way one has the opportunity to initiate several matches that have different parameters and group them, which is beneficial in the sense that one is able to separate the distinct matches. It will also provide an easier way of playing numerous matches with the same parameters having them stored in the same batch. Furthermore by concentrating chess matches into unique batches, it will become easier to look at a specific set of matches, this is especially useful in terms of the analysis.

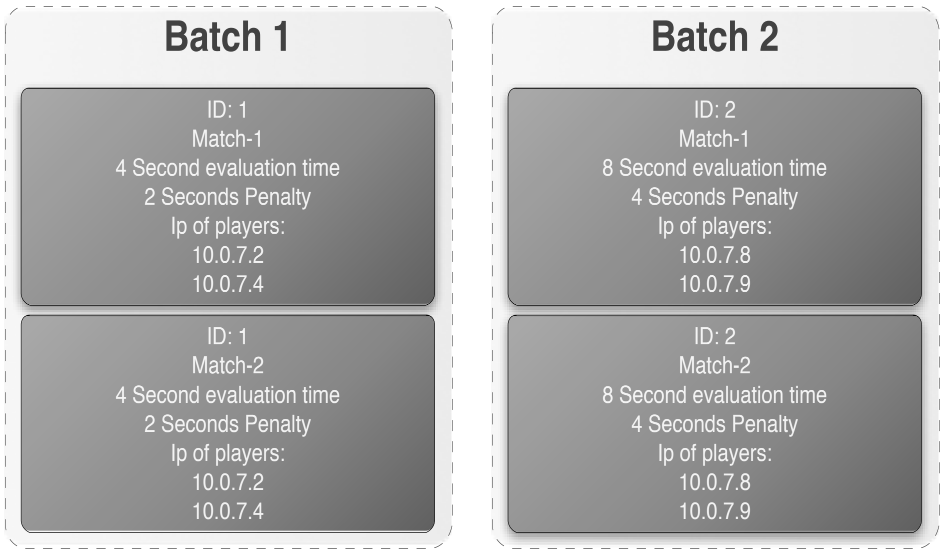


Figure 5.1: Illustrating the concept of batches, as a way of organizing matches.

Figure 5.1 depicts how a batch is thought out as a concept of storing chess matches. On the left having batch 1 with a set of parameters, and on the right another batch which has its own match settings. The figure illustrates how multiple matches may be associated with a single batch. Note: the database model will include the full description of a batch

5.2.2 Parallelization of matches

With the introduction of batches as a primary way of structuring the desired chess matches, and the subsequent data collected during their time of play, it will not only be possible to run one match, but numerous matches in parallel. This may be achieved by having several chess bots playing each other at any given time. Parallelization will make it possible to develop the framework in such a way that one may, instead of playing one single chess match and wait for it to end in order to start a new one, initiate several unique matches at the same time, seeing as they are apart of different batches and are uniquely identified. It will furthermore help reduce the time spent running chess matches considerably.

5.2.3 Chess bots

At this stage, the concept of a *chess server*, may be transmuted into that of a chess bot, of which there will be several. More specifically, The *chess bots* are in essence virtual machines with the latest version of Stockfish chess engine compiled on them. It will be necessary to configure the chess bots to listen on a specific port in order for it to be possible to orchestrate chess matches across the network between various chess bots.

One may also look at a specific chess bot as a chess *player*, seeing as one chess bot will be given the role as one of the two sides, either white or black. The reason why it is desirable to model the framework this way is that given the approach and ultimate goal of this research it will not be ideal to have for instance, one chess bot running multiple chess matches simultaneously. If the latter scenario was to be the case, it would impede the research to be conducted as it will be difficult to rely on the results. The train of thought is that if one chess bot is to be responsible for running several chess matches, one will have various chess matches competing for the resources of that particular chess bot. More over, the single chess bots capacity may be overbooked and the ending results may be inadequate as none of the chess matches will have had the opportunity to fully utilize the capacity of the chess bot itself.

As the research looks to identify how resource contention between multiple running virtual machines, consolidated on the same physical hardware, affects the overall quality of service of the applications running there, in this case a chess bot, it will not be desirable to have the chess bots capacity and performance diminished by internal processes, as the overall goal is to witness diminishing performance in light of the underlying physical resources being shared between multiple chess bots when running within the same environment.

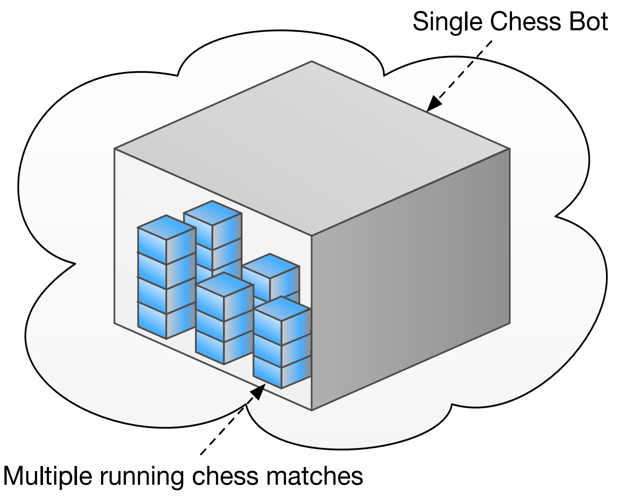


Figure 5.2: The above figure depicts the scenario where one would have one single chess bot running numerous chess matches.

The proposed solution of having one chess bot portraying one specific side during a game is a solution that may be seen as a alternative to that of the scenario just described. In its entirety the framework will be modeled in such a way that one chess bot will represent the white player, and accordingly one will have one chess bot representing the black player. By modeling the framework in such a way one has, in conjunction with batches, the opportunity to orchestrate a large number of individual chess matches at any give moment, and each chess bot will handle solely one side of the running match.

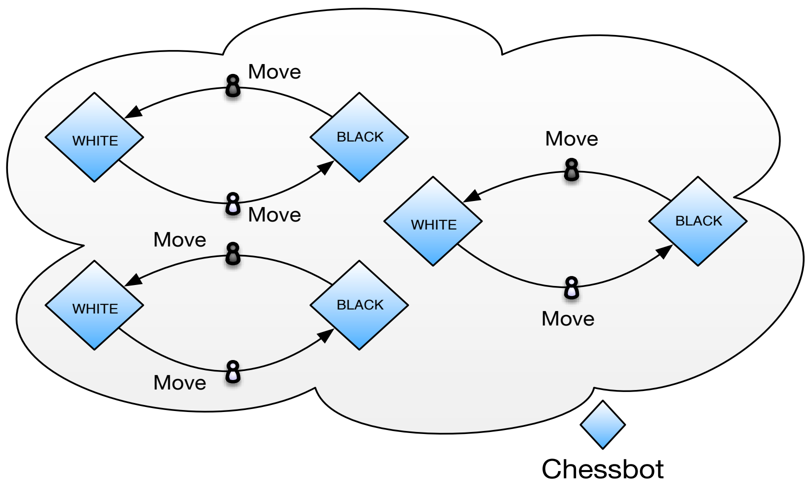


Figure 5.3: Illustration of the concept having white vs black player, each player being a separate chess bot in the cloud.

This way the chess bots will not have to account for other players or chess matches being played on that given chess bot. In this sense, it will be easier to look at the outcome of a single chess match in terms of how affected the end results are in terms of varying performance due to contention of resources in the environment where the chess bots will placed.

Figure 5.3 illustrates the concept of having a chess bot represent one side of a particular chess match. As one can see, this will enable the initiation of several chess matches at the same time, with each player being an independent chess bot. Although one may have the ability of running multiple matches at the same time if one adheres to this solution, it will be seemingly tedious having to initiate each batch of matches on the chess bots separately. Furthermore as the chess engine itself does not inhabit any feature allowing it to play across a network, a way to make sure that the chess bots are connected and communicating with each other and storage is of need. Therefore it may be convenient having a centralized communications node, in which will be able to communicate with the chess bots and storage accordingly.

5.3 The broker - Chess framework

As one is looking to orchestrate a large array of chess matches based in a virtualized cloud environment, and furthermore that these chess matches will be played out by independent chess bots, or "players", across the network, it will be beneficial to have a more centralized way of organizing the communication between the various chess bots and the storage module that will be at their disposal each match. As a result, within the final architecture, a controlling node will reside. Denoted *chess broker*. The proposed chess framework will be placed within this broker and the broker will be responsible for running the created modules within the framework in order to communicate with the chess bots and storage, and enabling the respective nodes to communicate with each other, although in an indirect manner. Furthermore the broker will be responsible for handling the analysis framework which will be addressed in section 6.3.

The intention of the chess broker will be that of initiating chess matches between the chess bots and convey the needed information between the two bots, white and black respectively, in order for the underlying chess engine to perform the wanted actions. The information conveyed by the broker will consist of data outputted from Stockfish during its evaluation of chess positions, and some of the information will be relayed back and forth between the two chess bots, such as the board representation, which player is next to move and furthermore the duration that Stockfish should run its evaluation of a given board. Furthermore the broker will be responsible for relying the data to the database, as this will be used by the analysis framework during the testing and resulting analysis.

The chess framework that is to reside within the broker will have to be designed in such a fashion that it will be possible to create the already mentioned batches, as they will be crucial in the game playing process. As a specific batch allows matches of the same specification to be stored within the same batch, the framework has to be modeled such that it will allow matches of the same specification to be repeated. This is a feature that will need to be adjustable, meaning, one is able to explicitly specify how many times a match of a given specification will be run. It will be beneficial if this is passed to the broker as a match, or set of matches are to begin.

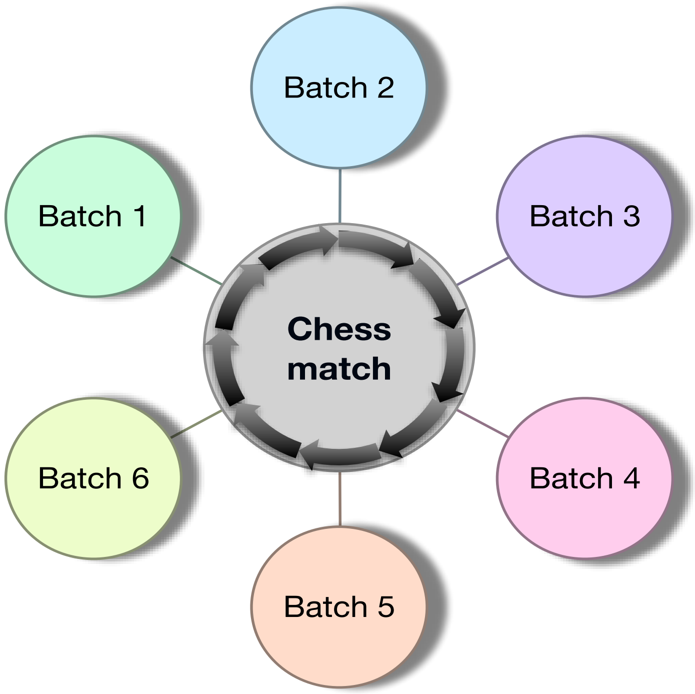


Figure 5.4: Illustrating the needed replay functionality of the framework

Figure 5.4 depicts intended the scenario in which a chess match with specific parameters, belonging to a batch is repeated. The intended match will be initiated with the following parameters;

- Which batch the match will be part of.
- Which chess bots are playing and their given side.
- How many times the given match scenario is to be repeated.

Another important factor to consider before initiating the development of the chess framework is that various scenarios that one may encounter while playing a game of chess. As a chess engine will continuously evaluate chess positions without considering the chance of obtaining a *remis*, which is essentially a draw in chess, the framework will have to take this into account when a match is initiated. A way one may claim a draw in chess is by threefold repetition, although in human chess, a player will have to ask for a draw when this situation occurs.

The threefold repetition entails that the same board representation occurs three times, the moves in which renders the same pattern on the board, need not come in succession in order to for it to apply towards the threefold repetition. As already mentioned the chess engine does not take this into account and this must therefore be implemented in the framework in such a way that it looks for repetition of board patterns. A way one may solve this is by looking at the FEN-string which represents the current board and the chess piece positions. If the same FEN-string is seen three times over the course of a match, one may call it a draw. The introduction of the threefold repetition rule may be beneficial in the sense that when chess matches are being played, if one of the sides are experiencing significant variation in performance, a match that might have ended up a remis, may actually turn out with a winner in the player who is not, or to a less degree experiencing loss in performance.

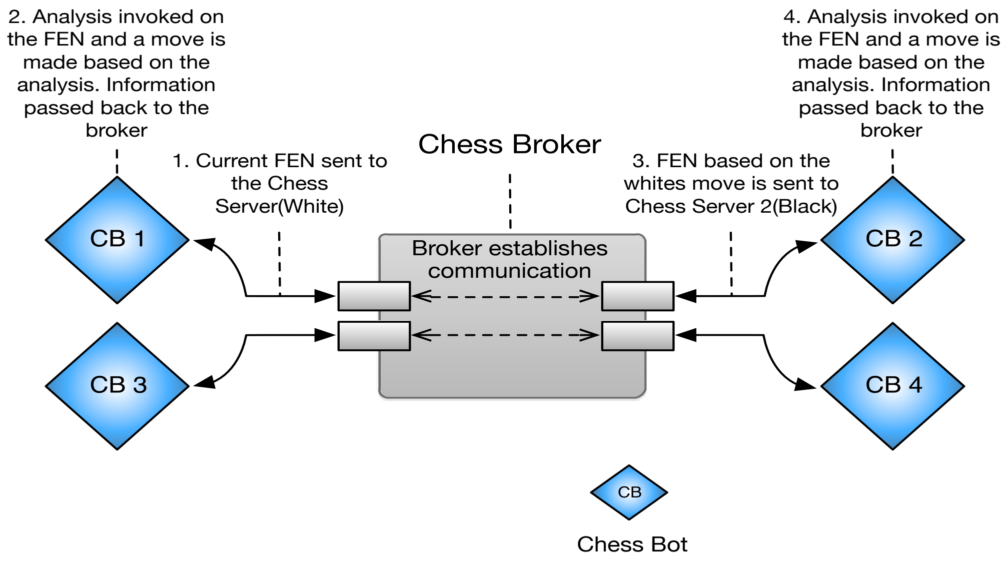

2. Analysis invoked on the FEN and a move is made based on the analysis. Information passed back to the broker
1. Current FEN sent to the Chess Server(White)
Chess Broker
3. FEN based on the whites move is sent to Chess Server 2(Black)
4. Analysis invoked on the FEN and a move is made based on the analysis. Information passed back to the broker
CB 1
Broker establishes communication
CB 2
CB 3
CB 4
CB
Chess Bot

Figure 5.5: The following illustration depicts how the intended chess broker may communicate with the chess engines. One may observe that chessbot 1(CB1) represents the white player, and chessbot 2(CB2) represents the black player.

Figure 5.5 shows a model of how the intended broker communicates with the various chess bots, where the broker initiates a match in which;

1. It sends the initial FEN-string(The initial Board) to the chess bot representing the white player.
2. The broker is then to pass instructions to the chess bot and it will perform an evaluation of the current FEN.
3. After the evaluation is complete, the broker initiates a move on whites behalf, retrieves a new FEN, and collects other significant data and passes it to storage.

4. The broker will then pass the new FEN, which is a result of whites move, to the black player. Which will be given the same instructions.

5.3.1 Match penalty

As previously mentioned there will be introduced a penalty to the matches in terms of the synthetic testing. This penalty may be either a fixed amount of seconds or a more uniform penalty in which a player is given a specific penalty but will also have a random generated amount of penalty seconds added to the sum. These added penalty seconds will decrease the time a given chess bot will be able to evaluate a specific FEN, meaning it will have less time to find the move considered to be the most optimal. This random penalty will never exceed the maximum evaluation time given when a match is initiated, and it will never drop below the imposed fixed penalty. E.g if the white and black player is to play a match where the evaluation time of Stockfish is set to 2000 milliseconds(2 seconds), and black is forced upon a penalty of 1 second, black will evaluate for a duration of 1 second + a random number of seconds. This may be 1000ms + 544ms = 1544. As opposed to white which will evaluate for the intended duration of 2 seconds.

5.4 Stockfish - What it provides in terms of data

The stockfish chess engine provides a significant amount of data during its evaluation of a current position. The longer the said evaluation is, the more data stockfish produces. Although the data increases in quantity, the fields of which the data is represented by, remains the same. Until Stockfish has reached the end of a given chess match, in which some fields change as a checkmate is expected within a certain number of moves.

Stockfish output
```
info depth 1 seldepth 1 score cp 19 nodes 48 nps 24000 time 2 multipv 1 pv e2e4 d7d5 b1c3 g8f6
info depth 2 seldepth 2 score cp 19 nodes 87 nps 43500 time 2 multipv 1 pv e2e4 d7d5 b1c3 g8f6
info depth 3 seldepth 2 score cp 19 nodes 141 nps 70500 time 2 multipv 1 pv e2e4 d7d5 b1c3 g8f6
info depth 4 seldepth 2 score cp 19 nodes 207 nps 103500 time 2 multipv 1 pv e2e4 d7d5 b1c3 g8f6
info depth 5 seldepth 2 score cp 19 nodes 308 nps 154000 time 2 multipv 1 pv e2e4 d7d5 b1c3 g8f6
```

Above one is an excerpt of an evaluation performed by Stockfish. When initiated, the evaluation function within Stockfish will run for the duration given in time or until a specific search depth is reached, as described in section 4.3.2. The specific data give by during the evaluation is;

- Depth - Depth can easily be explained as the depth at which Stockfish is evaluating at a given time. As one can see from the above excerpt, Stockfish outputs a line for each depth it has analyzed. The depth in which is searched is that of half moves, as mentioned in section 4.3.2. In reality a search depth of 2, is subsequently a search of one full move. In this lies that if the evaluation is initiated from whites point of view it would be an evaluation of one specific move for white and accordingly blacks counter move to the evaluated position.

- Seldepth - Seldepth,or selective depth depicts the depth at which Stockfish has done a more thorough search in specific branches of the search tree.

- CP - CP or centipawn is an important factor in chess as it is the score given to a specific move. E.g, if white moves and the CP results in "cp 20" it means that white has an advantage(Measured in 100th of a pawn, a pawn having a score of 1) of 0,2. Similarly for black it will be "CP -20"(Where the minus sign indicates black advantage).
- Nodes - Nodes is how many positions within the search tree Stockfish has searched. This field may accumulate to a substantial number if the engine is searching at a larger depth or longer evaluation time.
- Nodes Per Second(nps) - NPS is equivalent to Nodes above, only that it represents nodes searched per second.
- Time - Time represents the time Stockfish has spent evaluating a given depth.
- Multipv - Multipv or Multiple principal variation allows the display of several principal variation lines.
- Pv - Pv or principal variation is the sequence of moves that Stockfish considers best moves and are expected to be played.

Of the above mentioned, the type of data that will be important for this current research are the following;

- Depth
- CP
- Nodes
- Time
- Pv

The depth value will become valuable in terms of observing how for the various chess bots have been able to search at the time they are given, both with and without the impeding penalties given, and furthermore the contention they may exposed to during the live testing. The depth value is also an important factor when considering the analysis, as the chess referees will be set to run at specific depths in order to analyze the matches that have been played by the chess bots. This will be addressed further in section 6.3.

The centipawn value will be one of the most important values given by stockfish during a chess match, as it may be used to look at how well the two sides are performing under various scenarios. As the centipawn value denotes a score to every move made, it will be possible to use this in order to look at and create a cohesion between success and contention occurring within the environment. This will also be addressed further in section 6.3.

For the current research the nodes value provided by stockfish will be important as it may be used to identify how many positions stockfish is able to traverse during a given match scenario. In this lies that if a chess bot is experiencing diminishing performance as a result of contention within the environment, it may be possible to look at the number of nodes traversed and furthermore look for discrepancies in as to how many nodes it has managed to traverse relative to other matches played under other circumstances.

As an example, consider the scenario where one would have two instances of stockfish running within the same virtual machine. If one initiated the evaluation function within the two instances simultaneously, one would expect that they both shared approximately the same CPU capacity during evaluation. If one has run a single preliminary match beforehand with only one instance of stockfish running, with the same parameters, one would expect that the number of nodes traversed to be close to half of the traversed nodes during the match with a single instance running.

Data provided by in terms of time will be useful as it gives an indication as to how long stockfish has evaluated each ply depth during a single evaluation.

The data given by the principal variation field will be important as it will return what stockfish considers as the best move to make for a the currently moving side. This will be used when developing the framework as it will be possible to retrieve this value after each evaluation and make the framework(The broker) invoke the given best move for each player.

5.5 Intended run time scenario

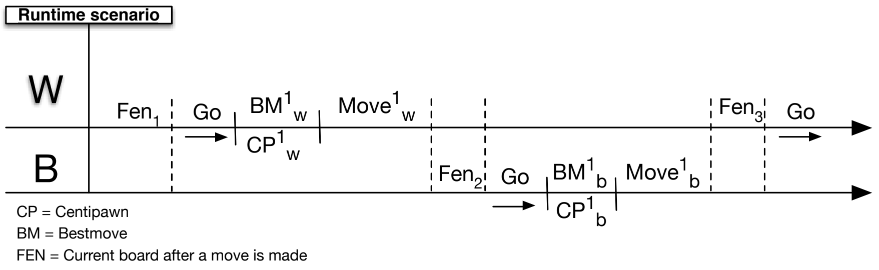
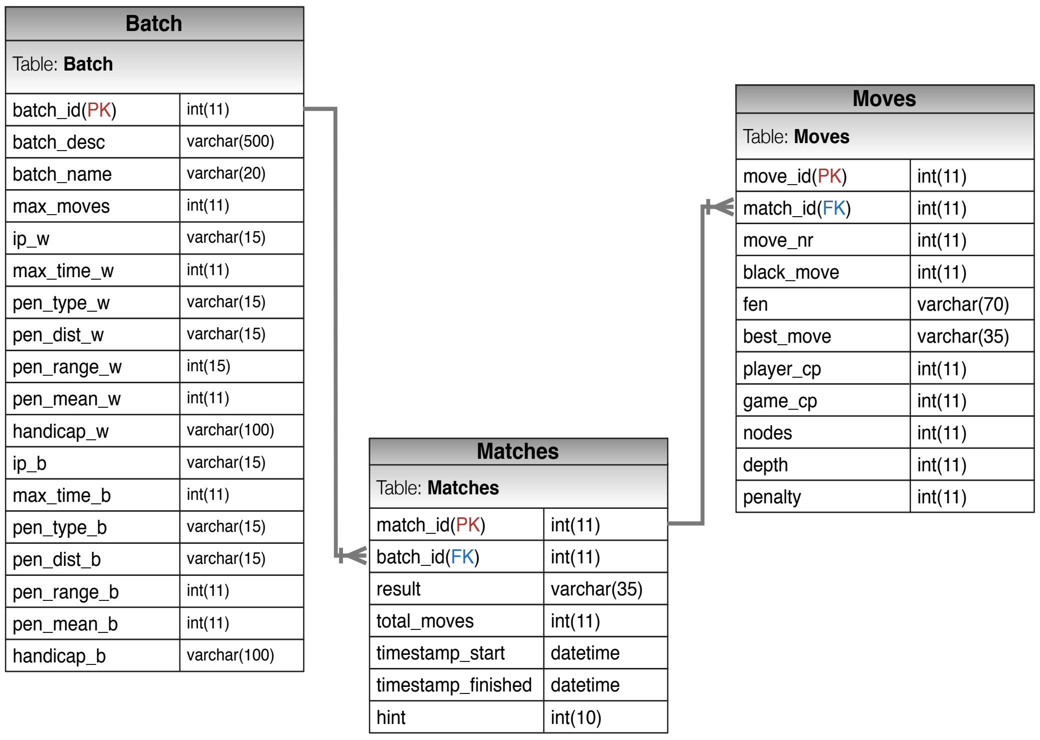


Figure 5.6: Showing a run time scenario of a chess match

Figure 5.6 depicts how the intended run time scenario as far as game play is concerned, penalties and other constraints aside. Of the figure one may see how the white player has the initial FEN set, and then the evaluation is triggered,*"GO"*, in order to find the most optimal move based on the said FEN is initiated. When the chess bot has finished evaluating its possible moves, white moves the piece in which has been found to be the best move(BM). The resulting best move also has an associated centi pawn(CP) value which will be stored for analysis.

5.6 Database model

As mentioned throughout the modeling chapter the broker will be retrieving and storing data as the chess matches progresses, and this section will describe the needed database model. The data stored will be crucial for the analysis framework as it will fetch the game data and have the opportunity to analyze it as it stored. The database structure proposed for the intended chess framework is as follows:



Batch

Table: **Batch**

batch_id(PK)	int(11)
batch_desc	varchar(500)
batch_name	varchar(20)
max_moves	int(11)
ip_w	varchar(15)
max_time_w	int(11)
pen_type_w	varchar(15)
pen_dist_w	varchar(15)
pen_range_w	int(15)
pen_mean_w	int(11)
handicap_w	varchar(100)
ip_b	varchar(15)
max_time_b	int(11)
pen_type_b	varchar(15)
pen_dist_b	varchar(15)
pen_range_b	int(11)
pen_mean_b	int(11)
handicap_b	varchar(100)

Matches

Table: **Matches**

match_id(PK)	int(11)
batch_id(FK)	int(11)
result	varchar(35)
total_moves	int(11)
timestamp_start	datetime
timestamp_finished	datetime
hint	int(10)

Moves

Table: **Moves**

move_id(PK)	int(11)
match_id(FK)	int(11)
move_nr	int(11)
black_move	int(11)
fen	varchar(70)
best_move	varchar(35)
player_cp	int(11)
game_cp	int(11)
nodes	int(11)
depth	int(11)
penalty	int(11)

Figure 5.7: The proposed database structure for the chess framework

Figure 5.7 displays the intended database model for this research. One may see that a specific batch may be related to one or more matches and that a given match will have one or more moves related to it.

Batch table

The first table one should consider is the *"Batch"* table, as a batch will be created in advance of initiating chess matches. An en entry in the batch table is what defines how a specific match, or a batch of matches will be played in terms of;

- Evaluation length(max_time_w/b)
- Which chess bots are playing the given match (ip_w/b)
- Which chess bot is to represent a given side. (pen_type_w/b)
- What type of distribution the penalty is (pen_dist_w/b)
- The range of which the penalty is expected to be within in in relation to the penalty mean(pen_dist_w/b).
- What duration of the imposed penalty is (pen_mean_w/b).

All of the batches will also be given individual names and have a description added to it in order to be able to distinguish them and as a way of summarizing the parameters given to each individual batch. Each batch will also be uniquely identified with an ID, as this will be used to connect the various matches to the correct batch.

Matches table

Accordingly there will be a need for creating a table for storing the relevant data in relation to a match in which an entry in the *matches* will need to contain;

- The ID of the batch the current match belongs to, this will be passed as a parameter when a match is initiated in order for it to be stored in the correct batch(batch_id).
- A match ID, as it will the connection between the match and the moves made during the match (match_id)
- What the end result of a match was, this may will depict the winner or if the result was a remis" (result)
- A column that stores the number of total moves made during a chess match.(total_moves)
- The timestamp of when a match was initiated and when it ended.(timestamp_start/finish)

Moves table

The last table needed in the database model is a table for storing moves and data related to each sides turn. En entry in the *"moves"* table will contain;

- A column representing a move ID for each move, as it will be vital when wanting to look at specific moves(move_id)
- A column representing the match ID of the match in which the moves belong.(match_id)
- A column representing what move number it is, this is indicated in half moves. Meaning that the move number will not increase before both players have moved. E.g (1. e2e4 1. e5e7) - (2. b1c3 2. b8c6) (move_nr)

- A column showing who the move belongs to, either white(0) or black(1). (black_move)
- A column showing the centi pawn score of each move, for each player(player_cp)
- A column showing the centi pawn value for the match. (game_cp)
- A column representing the number of nodes traversed in order to find that specific move(nodes)
- A column showing the depth at which the evaluation was made(depth)
- A column showing the sort of penalty the player had during the its evaluation. This will also display none-penalized move times.(penalty)

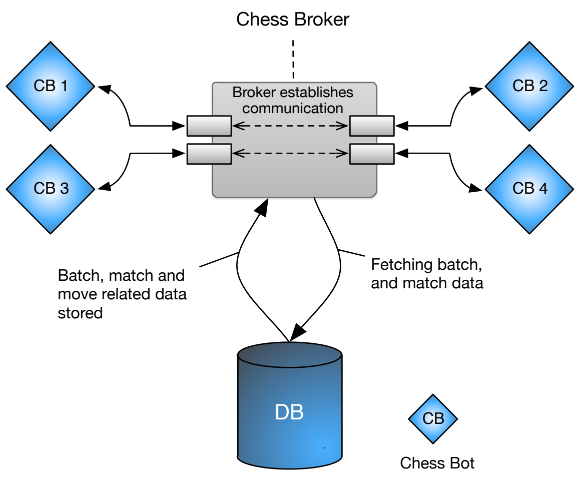

Chess Broker
CB 1
CB 2
CB 3
CB 4
Broker establishes communication
Batch, match and move related data stored
Fetching batch, and match data
DB
CB
Chess Bot

Figure 5.8: Illustrating how the broker is to store and collect data from the database

Figure 5.8 illustrates the concept where the broker communicates with the database and in which it stores all the data described above as each match is being played and whenever a move is being made.

5.7 Implementation of Chess Framework

5.7.1 Creating chess bots

The chess bots, and the proposed referee bots, are created in OpenStack from snapshots of predefined images. Each virtual machine that was to be created with the intention of being a chess bot was created from these images. The Stockfish chess engine is not created for the purpose of playing chess games between multiple hosts across the network, and it is therefore necessary to configure the virtual machines such that it is possible to connect to the chess bots over a specific port, and furthermore execute the Stockfish chess engine. In order to have the chess bot listen on a specific port for incoming requests, the *Xinetd* super-server daemon was used. Xinetd allows one to create services that listens on a specific port, and when an incoming request is handled, Xinetd launches the appropriate server connected to that specific service. In the case of this thesis, and in regards of the chess bots, a chess server script was implemented on each chess bot and was run whenever a request was made to the specified port.

Xinetd.d chess service

```
# default: on
# description: An RFC 863 discard server.
# This is the tcp version.
service chess
{
 disable         = no
 server          = /home/player/chess_server.sh
 socket_type     = stream
 protocol        = tcp
 user            = player
 wait            = no
 port            = 3333

}
```

The above code displays how the Xinetd.d chess service was implemented. The service points to that of */home/player/chess_server.sh* which is the script responsible for initiating the Stockfish chess engine whenever a request is made to port 3333.

Chess Server Script

```
#!/bin/bash

LOG="/tmp/chess-server.log"

/bin/echo "Start \$(/bin/date)" >> \$LOG
/bin/echo >> \$LOG

/usr/local/bin/stockfish

/bin/echo  | /usr/bin/tee -a \$LOG
/bin/echo "Stop \$(/bin/date)" | /usr/bin/tee -a \$LOG
```

The above code is the implemented script responsible for initiating Stockfish. As one can see, whenever a request is received, the binary of Stockfish, which resides in */usr/local/bin/stockfish* is run.

Figure 5.9 illustrates how the framework within the chess broker interact with the chess bots. As a match is initiated, the broker will send the a request through a

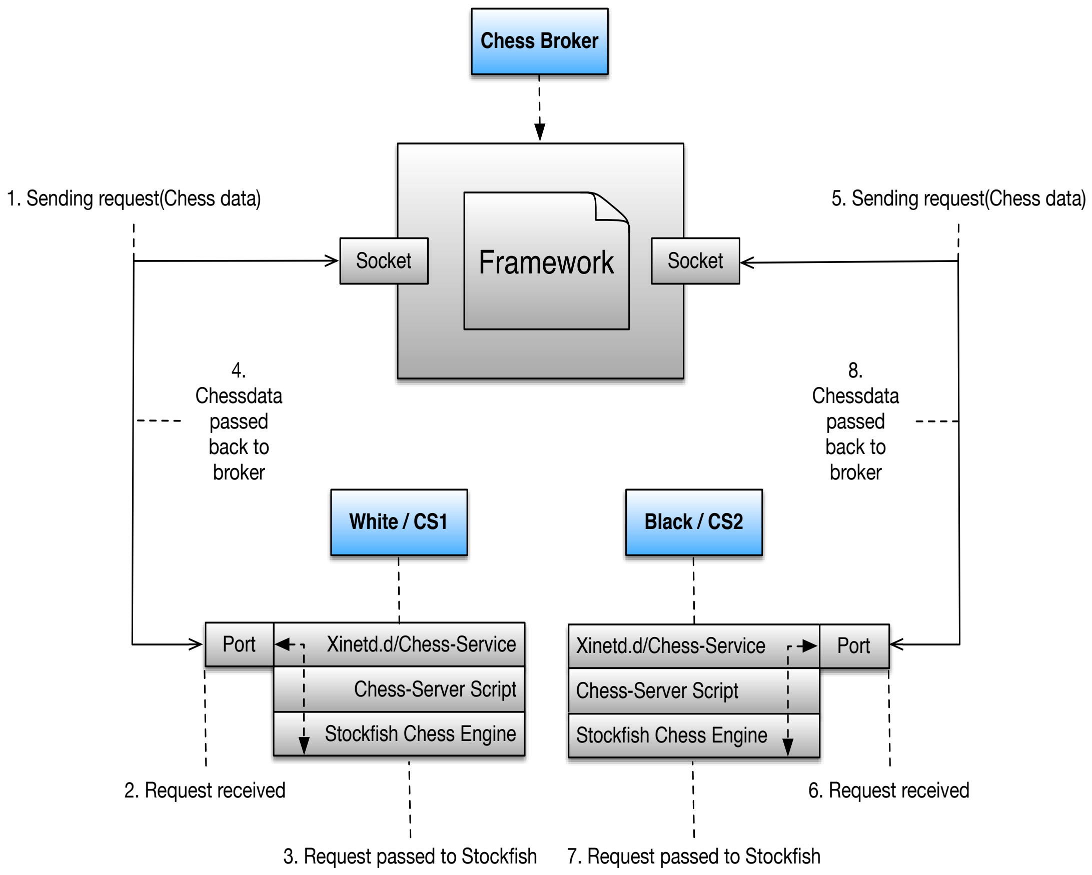


Figure 5.9: The process in which the broker communicates with the chess bots using Xinetd and chess server service

socket created by the framework script(1) to a specific port on the receiving chess bot(2). When the request has reached the chess bot, which may be that the chess bot is to initiate its evaluation of positions on the current FEN, the Xinetd daemon will make sure that the chess service will run the chess server script implemented on the chess bots. This chess server script will start the chess engine and the request sent by the broker will be run as a UCI command on the engine(3). The broker will then read the output and the data following the current evaluation, send some additional requests such as making the white player perform a move, and the data will be sent back to the broker, which will now send the updated FEN to the black player and perform the same actions.

5.7.2 The chess Framework - Script

In order to realize the chess framework described during the modeling section, a Perl script was created and is in its entirety the chess framework. *chess.pl*. As mentioned in chapter 4, the chess framework would rely heavily on automation in order to fully to satisfy the criteria in which entailed such as orchestrating a large array of chess matches being played simultaneously, communicating with chess bots, and furthermore be able to retrieve and store all the needed data produced by the various matches. The purpose of the chess framework script was to create various modules that followed up on these criteria and seamlessly intertwined them so that the only interaction needed with the framework would be when creating a new batch for organizing matches, and furthermore when initiating a set of matches with specific parameters that would be repeated until reaching the match limit in which was set in their corresponding batches.

5.7.3 Creating batches

The chess batches were created before a match, or a set of matches was initiated, as the purpose was to organize them in such a fashion that one would be able to play a set of matches with different parameters and store the related matches together, as it would ease the analysis process of the analysis framework. This section will look at how such a batch is created and furthermore the functionality behind it.

In order to create a batch one would use the following command:

Batch Creation Command
```
./chess.pl -B create -e "Batch description" -n Batch6 /
-P fixed:normal:100:0:fixed:normal:100:1000 -t 2000
```

The command displayed is used in order to create the wanted batches. As one may see, the command takes specific parameters in order to create a batch that is tailored after how one wants the matches to be played out. Especially for the synthetic testing some of the parameters are valuable as it determines how great of a penalty should be imposed on what player, if any, and if it is to be a constant penalty or more random in terms of withdrawn or added time. The parameters serve these functions:

- -B create - Signals that a new batch is in the creation
- -e Is the batch description parameter, in which one would describe the currently created batch.
- -n Will be the name of the currently created batch.
- -P takes a string of parameters which will be further explained in the next section.
- -t is the original amount of milliseconds the evaluation is supposed to run.

The fields of the string that is input to the script has following meanings:

- Noise type:Distribution type:Penalty range:Given penalty

The first field stating "Noise type", as may been seen as "Fixed" in the command, may also also say "uniform" in which fixed is a constant form of penalty(in seconds) and uniform being a constant penalty with some random noise generated and added to the already imposed fixed penalty. The field "Distribution type", represented by "normal" in the command, indicates what sort of distribution it is. Furthermore "Penalty range" field, indicated by "100" in the display above indicates within a certain range one may expect to find the penalty mean, if the match was run with uniform noise. The last field "Given penalty", or "2000" in the command excerpt is the actual fixed penalty given to the player in milliseconds.

Batch creation

Batch creation
```
If ( create, description, name, penalty string, time ) {
  run function to parse input string ->
    (split input string,
    return data to batch creation)

  connect to database
  insert provided values into database
  finish
  else { End batch creation as based on wrong or
        missing parameters
        finish
   }
}
```

When the framework is initiated on the broker with the broker creation command the framework will check the if the correct parameters have been issued, unless it will not create a new batch. Furthermore the framework will take the inputted penalty string and run it through a function that will split the string correctly, and together with the input values accompanying the other parameters, they will be inserted into a new entry in the batch table.

5.7.4 Initiating matches

When initiating a specific set of matches the following command is used:

Run match command
```
./chess.pl -p play -B 8 -x 30 -w 10.0.7.8 -b 10.0.7.9
```

The command displayed is the command that is used by the framework to initiate a set of matches. The command used to initiate a match, or set of matches takes a couple of parameters in order to correctly associate chess bots to each side of a match and furthermore how many matches are to be played. The parameters are the following:

- -p play - indicates that match is to be initiated
- -B will be the ID of the matches corresponding batch
- -x Tells the framework how many times the specified match is to be repeated
- -w IP of the white player
- -b IP of the black player

As the following command is issued, the framework will initiate the matches and they will play out without any form of interaction, only handled by the broker.

```
Match initiation
If ( play, BatchID, Numb of matches, whiteIP, blackIP ) {
  Connect to database
  Update batch table with player IPs where batch ID is
  that of the one supplied.

  while ( match is going ) {

  call run_match() function
  }
  finish
}
```

5.8 A game - Through the eyes of the broker

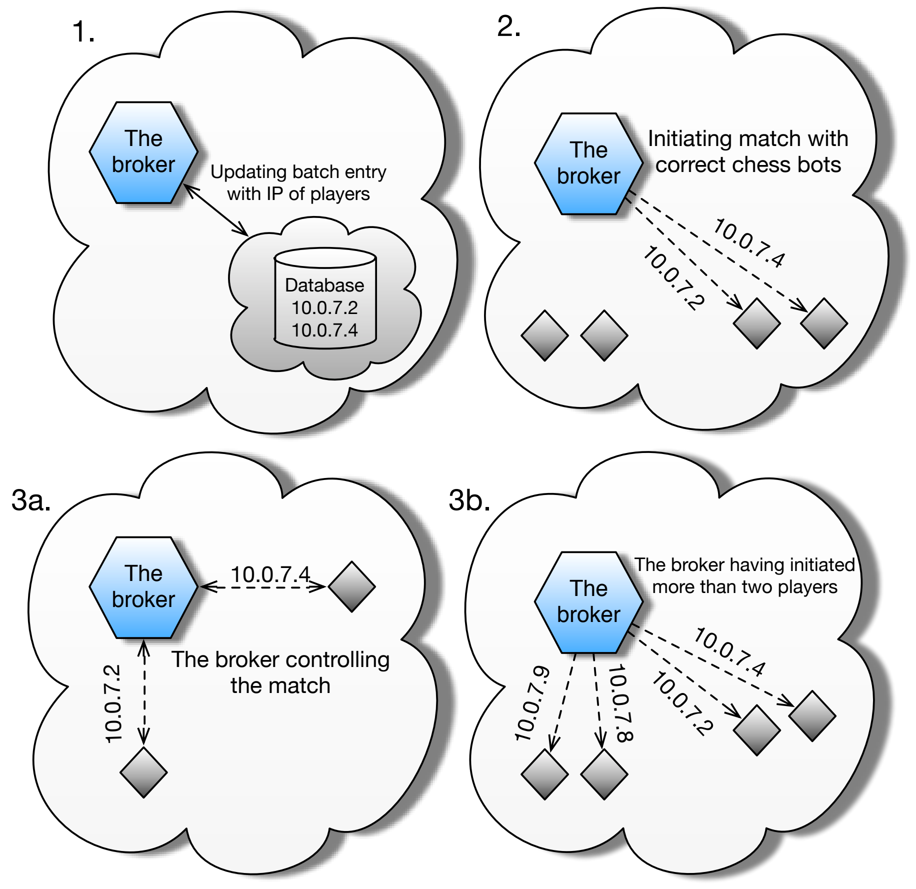

1.
The broker
Updating batch entry with IP of players
Database
10.0.7.2
10.0.7.4
2.
The broker
Initiating match with correct chess bots
10.0.7.4
10.0.7.2
3a.
The broker
10.0.7.4
The broker controlling the match
10.0.7.2
3b.
The broker
The broker having initiated more than two players
10.0.7.4
10.0.7.2
10.0.7.9
10.0.7.8

Figure 5.10: This illustration depicts the scenes where the broker starts a match, initiates the players and spawns more chessbots.

After the wanted batches are created a match is ready to be started. Figure 5.10 outlines a game play scenario seen from the brokers point of view. When a match is about to start, the framework within the broker will first update the batch to which the match belongs with the chess bots IP addresses, this is done in order to have control over which of the chess bots have played the different matches. the broker will then initiate the match between these two players and will keep control of it until all of the matches for the specific match is completed. Simultaneously, as a set of matches between two chess bots are underway, it is possible to launch more matches with other chess bots, which may be noted in the last frame of the figure.

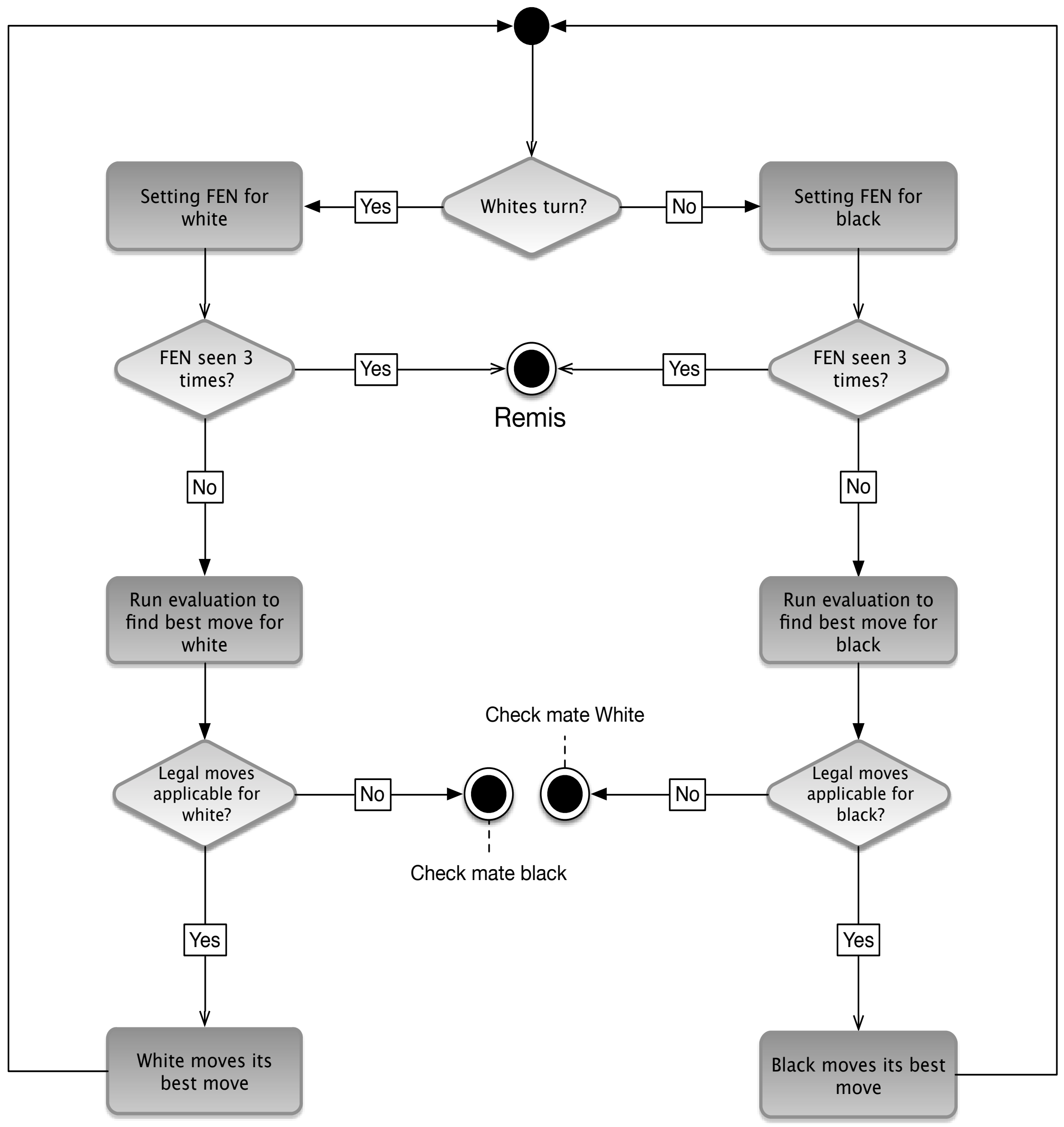

Whites turn?
Yes
No
Setting FEN for white
Setting FEN for black
FEN seen 3 times?
Yes
Yes
FEN seen 3 times?
Remis
No
No
Run evaluation to find best move for white
Run evaluation to find best move for black
Check mate White
Legal moves applicable for white?
No
No
Legal moves applicable for black?
Check mate black
Yes
Yes
White moves its best move
Black moves its best move

Figure 5.11: Flowchart of how a match is played out from its initiation and the possible outcomes of a match.

Figure 5.11 depicts a game scenario from the viewpoint of the chess broker. In order to fully appreciate how a match is played out by the broker, one may view a simplified version of the overall construction of the framework.

Game initiation

When the chess broker initiates a match between two chess bots, it will check for which of the chess bots are to make a move, in which it will look at which chess bot has been appointed white player, and this chess bot will automatically be set to move first. When white has made the initial move, the broker will check which player is next to move, and black will then be appointed the player next to move. In the event white player was a chess bot given the IP address 10.0.7.2 the mentioned scenario may be viewed as the following:

Who is to move?

```
While ( match is going ) {
  white player = 10.0.7.2
  call run_match()
  Whites turn finished -> New round

  if ( player is white ) {
  then switch to black )
  else { player is white ) }

  }
}
```

Setting FEN

As the broker has checked whether the player to move is white or black, the broker will set the current FEN for the chess bot to analyze. As mentioned the FEN represents the current board and its piece positions, as well as indicating which player is expected to move on the current board. The FEN which the broker will set is either the initial one given when a new match is initiated, or a FEN that is the result of one of the participating parties having moved a piece on the board.

Setting the FEN

```
While ( match is going ) {
   within the run_match function (
   initiate socket to current player
   set position fen 8/8/3k1p1p/5K2/R2P2P1/8/8/5r2 w 11 42
   )
}
```

Checking FEN

Whenever a FEN is set, the framework within the broker will perform a check in order to verify if the currently set FEN has occurred 2 times before during the ongoing match, if so, the rule of threefold repetition will be invoked as the FEN currently set will be the third occurrence. The result will be that the match is concluded in a *remis*, meaning a draw between the white and black chess bot.

Checking for threefold repetition

```
While ( match is going ) {
   within the run_match function(
   fetch current FEN from current match

   connect to database
   count number of occurrences of current FEN
   in database
   If ( fen is seen 2 times already ) {
     End match in remis
     finish match
   }else{ continue match }
   )
}
```

Finding the best move - Evaluation

After a specific FEN has been set, the broker will make the currently playing party(e.g white) start the evaluation of the current board in order to find the best possible move for the player to make. The duration of the evaluation of a chess bot, may either have a fixed number of seconds it is to run, or it may have an impeding penalty in which the evaluation time is shortened by the amount of penalty given. As for the synthetic testing, this penalty is forced upon the chess bot(s) in order to observe the outcome.

Running evaluation function

```
While ( match is going ) {
 within the run_match function(
 initiate socket to current player

  call calculatePenalty(
  connect to database
  fetch penalty given at match start

   If ( Penalty equals fixed ) {
   original time - fixed penalty
   return calculatedPenalty

   }elsif ( Penalty equals uniform ) {
   original time - fixed penalty
   calculatedPenalty + random noise
   return calculatedPenalty
   }
  )
 )
}
```

Making a move

As the evaluation finishes the broker will make sure that currently playing party moves the best move found based on the just run evaluation. In the case where the evaluation of the current FEN does not return any legal moves, it is because the opposing player has check mated the current player. And as a result, the match will result in a win to the player in which has obtained a check mate.

Making a move

```
While ( match is going ) {
 within the run_match function(
 initiate socket to current player
 call check_move(
   if ( No legal move found ) {
   return end of game
   }else{ return best move }
 )
 If ( no legal move found ) {
   Opposing player wins
 }else{ player moves best move }
 )
}
```

5.9 The Database

The database that is implemented in the current architecture is the relational database management system MySQL. The database is implemented on a separate virtual machine as it was highly unwanted that the database may be exposed to the varying conditions when the solution will be implemented in a live environment.

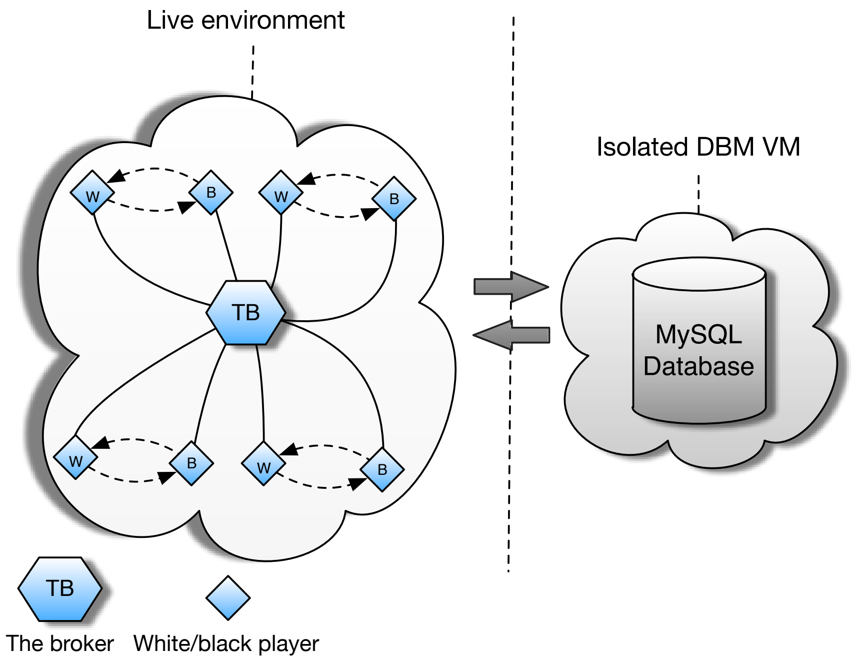


Figure 5.12: Database isolated from the rest of the machines

As it was desirable that the database was affected as little as possible during the tests, the database was moved out of the environment where the other virtual machines were running. As one may see of figure 5.12, the broker and the chess bots are running within their own secluded area of the architecture(located on another server), as they will be put under duress while running chess matches during the the proposed live testing. Furthermore as there are large sets of data being both passed and fetched from the database at any give time during a given match, and consequently even more data when multiple chess bots are playing simultaneously, it was highly desirable that the database was to be isolated from the unwanted noise and pollution created by the other virtual machines running within the testing environment. This way the database performed optimally at all times during both synthetic and possible live testing.

Chapter 6

Result - Analysis Framework

The previous result chapter featured the modeling and design of the chess framework and its intended implementation. The following chapter will introduce the analysis framework, and how it was modeled designed and implemented within the environment.

The proposed concept in terms of the analysis is that a functional and scalable analysis framework is to be developed. This analysis framework is to be implemented within the chess broker, along side the chess framework. The purpose of the proposed analysis framework is to utilize *referee bots* for analyzing the already played through moves and matches done by the chess bots. As to have the analysis be effective and efficient, there are several criteria that has to be considered when developing the analysis framework.

- How to measure success in a game
- A way of organizing the analyzed data.
- Creation of referees.
- Functionality for analyzing multiple batches.
- Creating a database model for the analysis.

6.1 Measuring success

As one of the overall aims(K1) is to map possible contention bound to virtual machines to that of success in terms of chess matches, one has to first define what success may concern. The concept of success is not easily identified or measured based on a game of chess, as one does not have concrete identifiers pointing directly to what concedes as success. Therefore, defining a set of surrogate variables is in place in order to measure the rate of success. A surrogate variable can be viewed as a measurable variable used in place of something that cannot be directly measured(e.g a chess match).

Concerning the aim of establishing a relationship between contention and success rate, two variables are of interest in order to define success, one of which is the centipawn value produced by the evaluation function of the chess engine. More specifically the centipawn is a score value that roughly corresponds to one hundredth of a pawn(2.3.3). This score is given to a move made during a game of chess, in order to display the advantage of a position, and is essential in chess engines when evaluating a chess match. Success rate may be broken down to that of success of individual moves. The thought of looking at isolated moves is based on the notion of looking at how well the evaluation of a given FEN is from a referees point of view, in contrast to that of the chess bots.

Once the chess bots are running or have completed an array of matches and the data from the subsequent moves have been stored, a number of chess referees, will fetch a selection of the data and analyze it in an effort of achieving a higher rate of success than that of the chess bots. And the aim being that a comparison of the centipawn value given by the chess bots and that of the referees will be conducted, in order to look at the possible differences in the centipawn value. This comparison will be more reliable if the referees are given more room for evaluation and being isolated as much as possible from the noise within a live environment. As it will not be optimal having the referees in an environment where they may be exposed to possible contention and have varying degrees of resource capacity, as the goal is to establish a sense of proof that contention in fact impacts the overall quality of service of the chess matches played by the chess bots. Based on this one may denote *cp_loss* as the variable of which will represent the loss in quality of a given move made by the chess bot. Based on the success rate in terms of the centi pawn value. More specifically the value represented by cp_loss may be seen as the analysis frameworks way of identifying whether the referees produced a higher quality move during the analysis compared to the move found by the players during the original match. This will be further explained in section 6.3.1.

The following mathematical formula serves as how one may look at the quality of a move:

$$CP_{loss} = CP^{1}_{otimal} - CP^{2}_{proposed}$$

The second variable of interest when concerned with defining success, is that of nodes, and nodes over time. Nodes may be defined as the number of positions the chess engine is able to evaluate within the span of a given time frame or a certain search depth. Nodes over time reveals the speed of the chess engine as it gives an indication of how many nodes it has managed to traverse. Although nodes over time may represent the speed of the engine, it may be used to look at how many nodes it has managed to traverse in a given scenario, as the engine relies on the CPU to traverse such vast amount of nodes. As a result, an introduction of noise or variation in CPU performance during evaluation may impact the number of nodes it manages to traverse. Nodes over time may be denoted with the following expression(A bit uncertain as to how mathematically formulate this):

$$\text{Nodes over time} = N/t$$

6.2 Organizing the analysis

6.2.1 Analysis batches

As with the chess framework, the proposed concept of organizing data in batches is also valid for the analysis framework. The data stored within these analysis batches, will be based on the data produced by the already played, or currently running chess games. The proposed solution is having the analysis framework analyze a specific batch of matches and then store the processed data within its own analysis batch. This way it will be possible to separate the analyzed data on specific matches and also which batch they originally belonged to. This will be helpful as one will want to look at independent matches with different parameters and characteristics. An analysis batch will furthermore act as template for how the analysis should be run as it indicates what parameters the analysis should be run with.

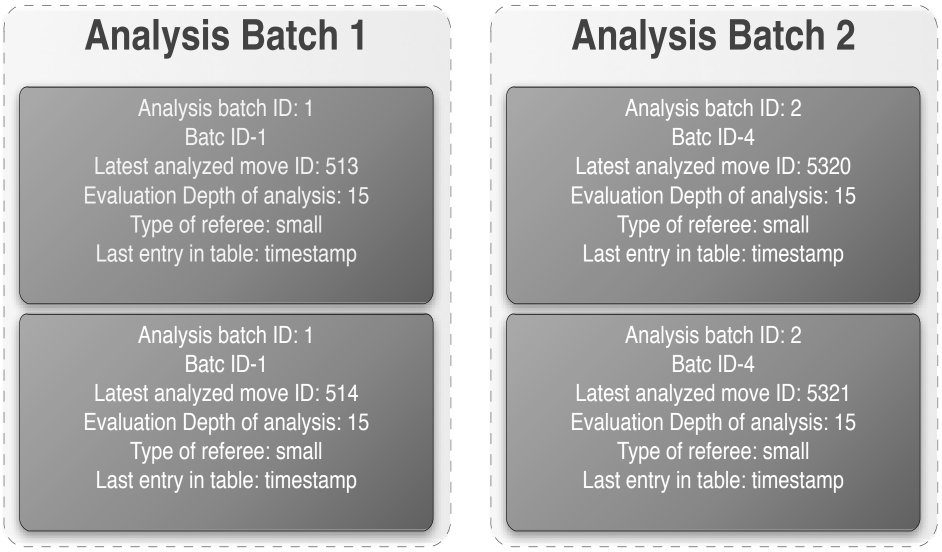

Analysis Batch 1
Analysis batch ID: 1
Batc ID-1
Latest analyzed move ID: 513
Evaluation Depth of analysis: 15
Type of referee: small
Last entry in table: timestamp
Analysis batch ID: 1
Batc ID-1
Latest analyzed move ID: 514
Evaluation Depth of analysis: 15
Type of referee: small
Last entry in table: timestamp
Analysis Batch 2
Analysis batch ID: 2
Batc ID-4
Latest analyzed move ID: 5320
Evaluation Depth of analysis: 15
Type of referee: small
Last entry in table: timestamp
Analysis batch ID: 2
Batc ID-4
Latest analyzed move ID: 5321
Evaluation Depth of analysis: 15
Type of referee: small
Last entry in table: timestamp

Figure 6.1: Illustrating the concept analysis batches to store analyzed data.

Figure 6.1 depicts a set of proposed analysis batches in which an analysis batch has an ID of its own, and the analysis batch should also inhabit a field indicating which chess batch it is currently set to analyze. Furthermore it will be beneficial to update the analysis batch with the latest chess move currently analyzed. It will also be necessary to indicate at which depth the evaluation the referee should analyze, and what sort of type the referee is in terms of capacity. Lastly a field indicating when the analysis batch was last updated is wanted.

6.2.2 Referees

The mentioned referees or referee bots are in essence, like the chess bots described in section 5.7.1, virtual machines with the latest version of Stockfish compiled on them. They are also configured such that the broker may communicate with a referee through a specific port. A key difference that will set the referees aside from the chess bots is that it will be possible to configure the referees to run at different capacities. The reason for this possibility is that one is interested in identifying exactly how much better the chess bots might have played in the off chance that they are not affected by impeding performance variation and contention within the environment. By having referees isolated and being able to play in a significantly less noisy environment, with increased capacity, one may be able to obtain results that one may consider to be how the chess bots should have been performing.

6.2.3 Parallel analysis

As one looks to analyze the data given by the chess matches, it would be optimal having several referees analyzing game data at any given time. As one match may produce a significant amount of data it will be time consuming having only one referee handling the analysis. One may consider the scenario where one may have a batch which contains 100 chess matches, if one then consider that each match may contain between 60-190 moves, it amounts to a total of 6000-19000 moves for a single batch of matches. As a result, one is interested in running several referees in parallel with the opportunity of analyzing both the same and separate batches simultaneously, as this will increase the rate of analyzed moves considerably. Moreover, as the proposed referees may be created with different capacities, it is desirable being able to initiate an analysis on a batch with two distinct referees as one may then compare the analysis of the two referees.

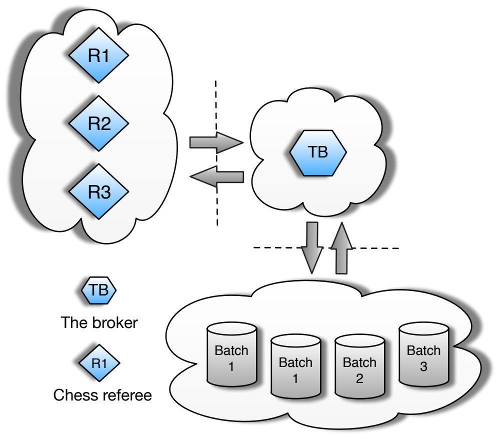


Figure 6.2: Illustrating the wanted parallel analysis scenario

Figure 6.2 illustrates the wanted functionality in that the broker is responsible for fetching the wanted data from several batches stored in the batch table within the database, and then distributes this among the referees that has been initiated. In this proposed scenario it will be necessary to create a solution such that the broker will know to which referee it should distribute the data retrieved for analysis. Hence a similar functionality is needed where one is able to specify to the broker which referee is to handle the analysis.

6.3 The Broker - Analysis Framework

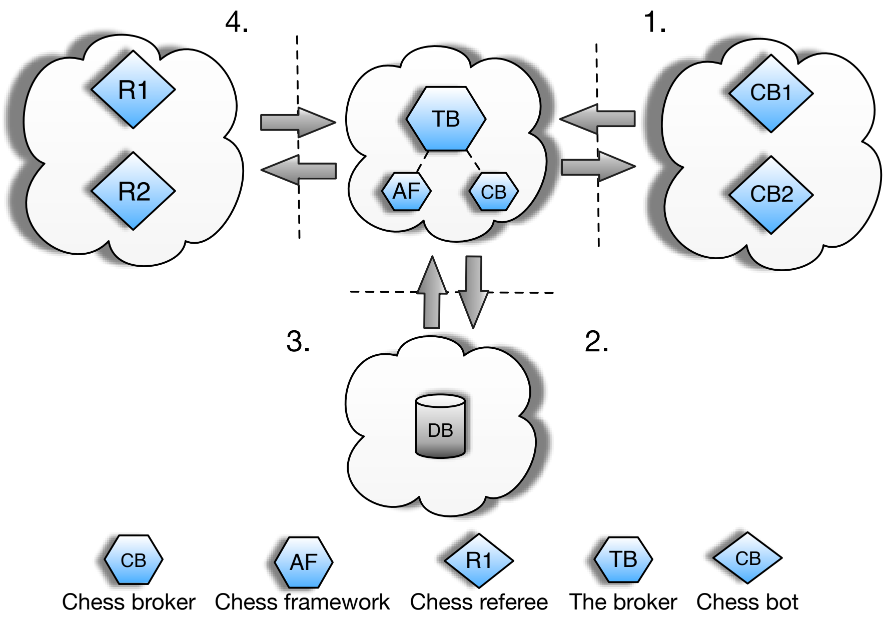


Figure 6.3: An overview of the infrastructure in terms of the broker communicating with referees, chess bots and the database

As mentioned in section 5.3 the broker will be responsible for handling the chess framework, which in itself is the main tool for organizing and orchestrating chess matches on a large scale. The broker will be responsible for the creation of chess batches in which the related matches will be stored and subsequently store all of the data that is produced by each significant match. This storing of data is done in such a manner that the analysis will be eased. In this lies that when wanting to analyze the stored data, and if one wants to analyze a particular set of matches that were run or are running with specific parameters, one may simply fetch the wanted batch and with it comes the related move data.

As the chess matches are initiated, another proposed framework, the *analysis framework*, will either be run simultaneously along side the chess framework, or may be invoked after a set of matches have been played. The broker will, simultaneously as it is orchestrating the initiated chess matches, be responsible for handling the analysis framework and its intended functions for analyzing chess data. As with the chess bots, the broker will furthermore be responsible for initiating the chess referees as they will handle the execution of the analysis itself.

Figure 6.3 displays an overall overview of the intended architecture where the chess broker will handle both of the frameworks and the communication between the several referee and chess nodes. It furthermore depicts how the broker will communicate with the the database and distribute the data between the respective nodes. To appreciate the architecture in its entirety, a more detailed description of figure 6.3 is favorable.

1. The broker will first initiate chess matches between various chess bots, in this case represented by chess bot 1 and 2.
2. The data produced by each chess bot in terms of move and match related data will be stored in the database by the broker.
3. As the matches are undergoing, or after they are finished, the broker will initiate the analysis framework and fetch move data from the database.
4. The broker will then distribute this data to the various referee bots that will be conducting the analysis. The analyzed data will subsequently be stored in the database.

In order to achieve the wanted level of parallelism described in the section concerning parallel analysis, a specific function is needed that allows for fetching data from the same batch within the database simultaneously in such a way that one may avoid race conditions. If such a solution is not implemented, one may experience having multiple referees analyze the same moves more than once. This scenario would not be trivial as it would be a waste of resources running an analysis on the same move more than once, and it will fill the database with multiple duplicates.

Furthermore will the proposed analysis framework require functionality that allows one to fetch the best move made by the chess bots and re-iterate this move during the analysis, only now on a significantly deeper evaluation in order to see if the chess bots chosen move was indeed the best move to make. The phases of the analysis will be;

1. Fetch the FENs from an original chess match played by the chess bots
2. Run an initial evaluation on the FEN in order to find the proposed moves that the player may have invoked during the original match, and its centi pawn values.
3. Invoke the best proposed move on the currently set FEN(Which is fetched from the original match).
4. Run an evaluation on this FEN in order to find the optimal move, and subsequently the optimal centi pawn value.
5. This centi pawn value given by the best proposed move and the optimal move will be grounds for calculating the quality of a move made during the original match, in which will be further explained in section 6.3.1

6.3.1 Quality of a move

As briefly mentioned during the introduction of this chapter when elaborating on the matter of measuring success(6.1), one is interested in looking at the potential loss of quality in a move made by a chess bot. Based on the success rate measured in centi pawns. In this lies that during the analysis, the referee bots will be fetching move data from the already played through chess matches and replaying the given scenarios. During the analysis the proposed referees are to analyze the fetched move data on a depth that should always exceed the evaluation depth of the player in order to obtain meaningful results. The denoted cp_loss value will be an important factor during the analysis, as it will be used to indicate whether the chess bots during the original match have in fact made a move that would be seemingly poorer to that of a move found during the analysis, as this would have been the optimal move to make.

During the analysis of a given move, if the best move found after the evaluation of a given FEN, by a chess bot during the originally match, is the same as the move found during a referee bots initial analysis(proposed move), one would expect the cp_loss value to be 0. In this lies that the move found by the player, at e.g depth 9 is of the same quality as the move found by the referee during an analysis at for instance depth 15 or 20. Meaning the success of the moves were equal. Although in the case where the move found during the analysis differs from that one found during the original match, one would expect that the move made by the player, be of lower quality. The move found during the analysis would seemingly be stronger because of its deeper evaluation.

Based on the mathematical formula(6.1), one may denote the proposed move as the move found during the initial analysis run by the referee bots. The initial analysis may be considered the scenario where the referees finds the best move to make for a given FEN retrieved from the match in which it is analyzing. This will be considered the proposed move, meaning the move that the chess bot should have made during its original match. And it has related success rate, in centi pawns.

In order to calculate a possible loss in quality, the proposed move found during the initial analysis should be invoked on the current FEN, meaning the proposed move should be moved on the current board. As the move is made a new FEN is generated, and an evaluation should be run in order to obtain what the optimal move would be. The optimal move found, will give a related success rate in terms of a centi pawn value, and it is the deviation of this optimal rate of success to that of the success rate of the proposed move that the cp_loss will be calculated. And it tells us something about the loss in quality of the move made by the chess bot.

6.4 Intended run time scenario

In figure 6.4 one can observe the intended run time scenario for a referee in conjunction with the running chess matches. The game scenario will run its intended course but as the matches are underway, the referee will start its analysis.

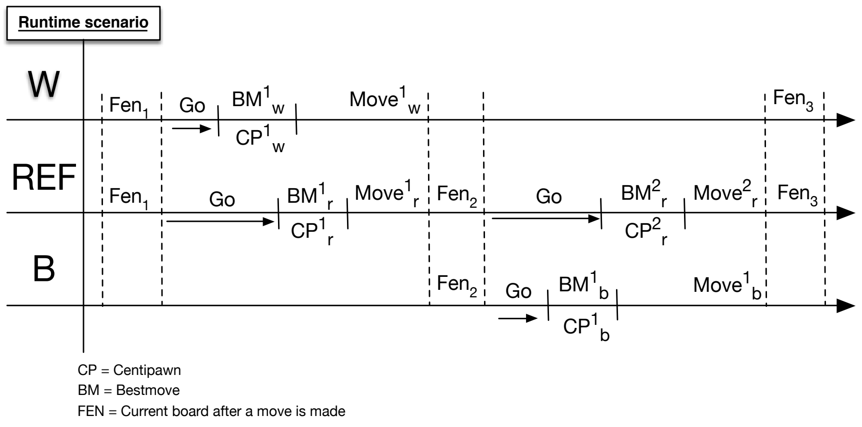


Figure 6.4: Showing a run time scenario of a chess match with a concurrent analysis of the ongoing or stored matches.

Concurrently as the white player has made its move, a separate chess referee will invoke its analysis of the move just made. The chess referee will be analyzing the move deeper than that of the chess bots and therefore the evaluation process will take more time.

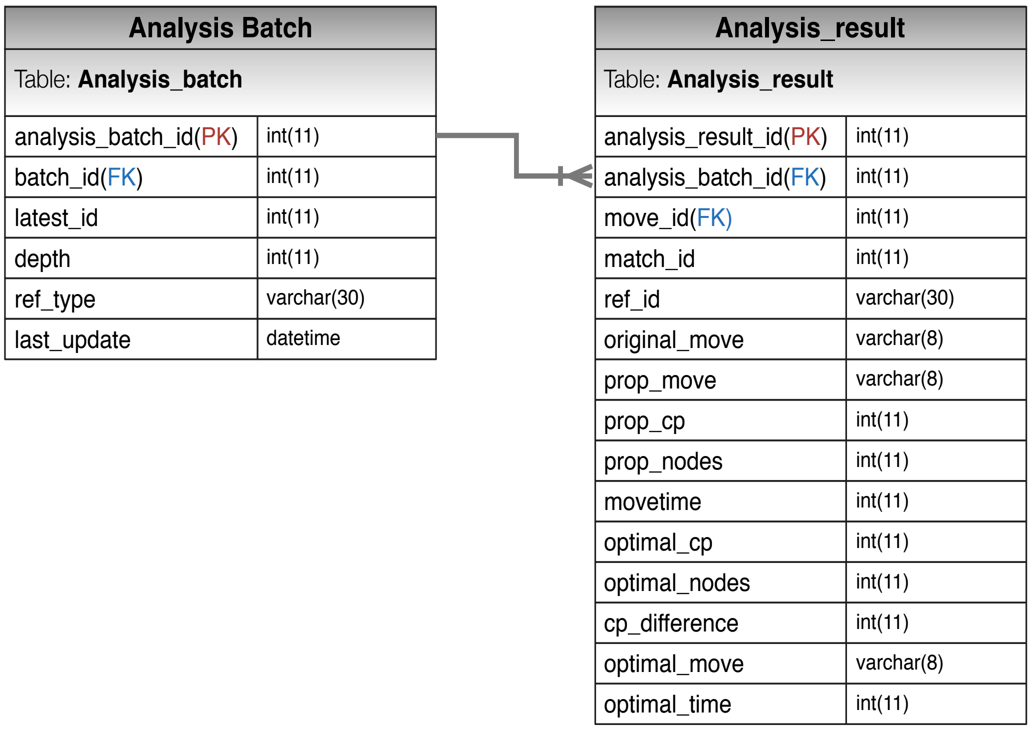
6.5 Database model

The analysis framework will, as mentioned during the modeling of the chess framework, have the opportunity to fetch data from the running matches as they are running. And during the introduction of this chapter a solution for storing the analyzed data in organized fashion using analysis batches was proposed. The proposed database structure may be seen in 6.5. An analysis batch may have many analyzed moves associated with, these moves will belong to a specific batch specified in the analysis batch entry. The data stored in the results table will accordingly belong to one specific analysis batch entry in the database.

An entry in the analysis batch table is what will define how a given analysis will be conducted and by which referee. An entry in this table will contain:

Analysis batch

- A column defining the ID of the current analysis batch (analysis_batch_id)
- A column representing the ID of the batch that will be analyzed (batch_id)
- A column indicating which move id was last analyzed (latest_id)
- A column indicating at which depth the referee will be evaluating. (depth)



Analysis Batch	
Table: **Analysis_batch**	
analysis_batch_id(PK)	int(11)
batch_id(FK)	int(11)
latest_id	int(11)
depth	int(11)
ref_type	varchar(30)
last_update	datetime

Analysis_result	
Table: **Analysis_result**	
analysis_result_id(PK)	int(11)
analysis_batch_id(FK)	int(11)
move_id(FK)	int(11)
match_id	int(11)
ref_id	varchar(30)
original_move	varchar(8)
prop_move	varchar(8)
prop_cp	int(11)
prop_nodes	int(11)
movetime	int(11)
optimal_cp	int(11)
optimal_nodes	int(11)
cp_difference	int(11)
optimal_move	varchar(8)
optimal_time	int(11)

Figure 6.5: Proposed database structure for the analysis framework

- A column representing what type of referee will perform the analysis (ref_type)

- A column indicating when the analysis batch was last updated with new information (last_update)

Analysis result

Furthermore, each entry in the analysis batch table has corresponding entries in the analysis result table which will contain all of the analyzed data. An entry in this table will consist of:

- A column defining the ID of an entry in the analysis result table. (analysis_result_id)

- A column representing the ID of which analysis batch the analyzed move belongs to.(analysis_batch_id)

- A column representing the ID of which move is being analyzed (move_id)

- A column representing the ID of which match the analyzed move belongs to (match_id)

- A column representing the ID of which referee bot is currently analyzing the given move (ref_id)

- A column identifying the original move of which is currently being analyzed (original_move)
- A column identifying the proposed move based on the original move given during the analysis conducted by the referee. (prop_move)
- A column identifying the centipawn value of the proposed moves given by the analysis. (prop_cp)
- A column identifying the number of nodes traversed during the analysis of the proposed moves (nodes)
- A column representing the time it took conducting the analysis on the given depth. (movetime)
- A column representing the optimal centipawn value given by the analysis of the proposed moves (optimal_cp)
- A column representing the optimal nodes traversed given by the analysis of the proposed moves(optimal_nodes)
- A column representing the difference in centipawn value between the proposed centipawn value and the optimal centipawn value(cp_difference)
- A column representing the optimal move found by the analysis of the proposed move (optimal_move)
- A column representing the time the evaluation function spent finding the optimal move (optimal_time)

6.6 Implementation of Analysis framework

This section of the chapter will elaborate on how proposed analysis framework was created and implemented within the architecture, and it will furthermore detail certain functionality and concepts described in the previous modeling chapter.

6.7 Analysis framework - Script

In order to successfully meet the criteria described in the modeling section of this chapter, a perl script was developed. This script created, *analysis.pl*, is as with *chess.pl*, in its entirety the analysis framework. As with the chess script the analysis script would rely on automation in order for the referees to constantly be analyzing match data. The purpose of the analysis script was to serve as framework within the broker, in which the broker would have the responsibility of handling the communication between the various referees and the database during the analysis process. The analysis script, similarly to the chess script was to contain various functions that were neatly intertwined in order for the process of analysis to work seamlessly.

6.7.1 Creating analysis batches

The analysis batches was created before an analysis was initiated, and the purpose is as with the chess batches because it eases the analysis process in the sense that one is able to group the analyzed data together and easily extract the data one needs. Furthermore as the referees will analyze a given batch of matches with different specifications in terms of evaluation depth, it is optimal to have the analyzed data separated.

In order to create an analysis batch one needs to invoke the following command to the framework:

Analysis Batch Creation

```
./referee.pl -c create -b 3 -t m1.small -d 15
```

The command displayed above is used in creating an analysis batch. The command, like the one used when creating a regular batch, takes a set of obligatory parameters that will is used to create specific and independent analysis batches. The parameters serve the following purpose.

- -c create - Specifies that a new batch is being created.
- -b identifies which chess batch to analyze.
- -t Represents what type the referee analyzing the batch will be
- -d Represents at which search depth the referee is to analyze. This should always be more than the original players in order to achieve meaningful results.

Creating analysis batches

Analysis batch creation

```
If ( create, chess batch ID, type, search depth ) {

  connect to database
  insert provided values into analysis batch table
  finish
  else { End batch creation as based on wrong or
        missing parameters
        finish
   }
}
```

6.7.2 Initiating an analysis

When wanting to initiate an analysis process one initiates the analysis framework with the following command:

Analysis Batch Creation

```
./referee.pl -A analyze -r 10.0.7.5 -i 7 -d 15
```

The above will be invoked when initiating an analysis of a given chess batch. The parameters are as follows:

- -A analyze - Specifies that an analysis is to be conducted.
- -r specifies the IP address of the referee analyzing the chess batch.
- -i species which analysis batch the analysis should be conducted on.
- -d specifies the depth at which the analysis will be conducted and has to match the depth given in the analysis batch.

Initiating Analysis

```
Initiating analysis
If (analyze, analysis batch id, search depth,
    search depth) {

  connect to database

  while ( analysis going ) {
    call fetch_data() function
    call run_analysis() function
 else { Analysis not initiated due to wrong or
        missing parameters
        finish
   }
}
```

6.8 How an analysis is conducted

Once the needed analysis batches have been created, one can initiate an analysis. Of figure 6.7 one may see how an analysis is conducted. Of the illustration one may observe how the chess broker, as with the initiating of a chess match, first updates the analysis batch with the IP address of the respective referee bots that are to analyze the given batch. After the update, the broker will also fetch the first available move to analyze from the batch it the respective referee has been set to analyze. Next the broker will distribute the fetched move to the said referee for analysis. The broker will then control the analysis for as long as there are new moves to analyze. From the frame 3b, one may see how the broker initiates more referees to run simultaneously.

As noted during the introduction of this chapter(6), a criteria for analyzing multiple batches had to be met in order for the framework to function as intended, and furthermore that parallel analysis of a single analysis batch asks for a solution to deal with race conditions(6.3). This is needed as it is not desirable to have duplicated data in the database because of having the same move being analyzed more than once, furthermore it would diminish performance of the analysis. Based on this the analysis framework was created in such a way that it allowed locking of the various table rows in the database. This means that each time the broker

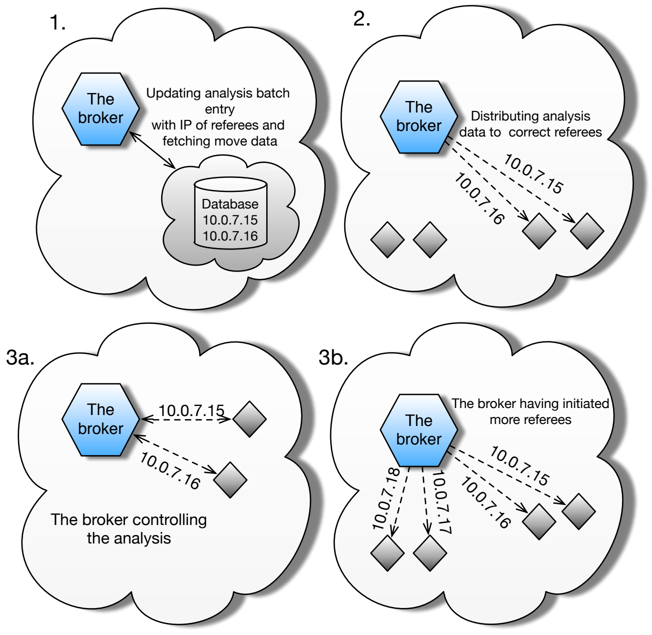

1.
The broker
Updating analysis batch entry with IP of referees and fetching move data
Database
10.0.7.15
10.0.7.16
2.
The broker
Distributing analysis data to correct referees
10.0.7.15
10.0.7.16
3a.
The broker
10.0.7.15
10.0.7.16
The broker controlling the analysis
3b.
The broker
The broker having initiated more referees
10.0.7.15
10.0.7.16
10.0.7.17
10.0.7.18

Figure 6.6: Illustrating how the analysis is conducted from the brokers point of view.

is about to fetch a move for analysis in the database, a check will be performed in order to identify which move was last updated, but the broker will not be allowed to this before another instance of the broker has finished fetching a move for another ongoing analysis. This way one is able to circumvent the race condition factor. This locking is possible by using a built in feature in the MySQL database that will lock a specific row when it is being read, keeping others from reading the same row. When the read operation is complete, it will unlock the table for further reading such that the next read operation may be permitted.

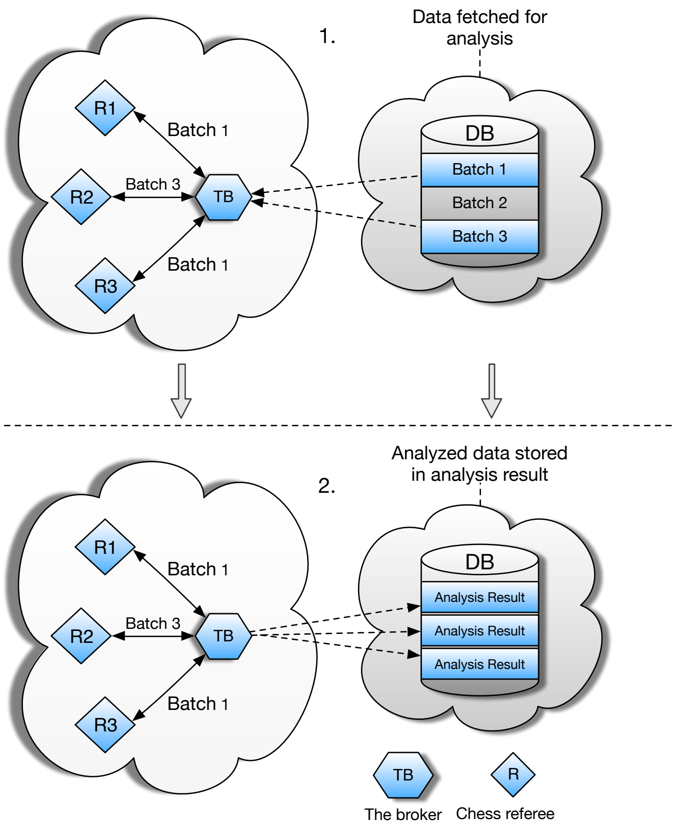


Figure 6.7: Illustrating how the referees interact with the database

Of figure 6.7, illustration 1, one may observe how the broker fetches data from two different analysis batches and distributes the data to the correct referees, as specified in the analysis batch. One may observe that two referees, referees 1 and 3 respectively, are analyzing moves from the same batch, and as mentioned in the last paragraph this would have prompted a race condition if the two were to fetch data from the batch simultaneously, but because of the locking of table rows in the database this is avoided.

If one further observes illustration 2, one can see how the analyzed data is stored within the database. The analyzed data will be stored in separate rows within the analysis result table.

Chapter 7

Analysis

The following chapter will look at data produced by the chess matches run by the chess framework and data analyzed by the analysis framework. The data in this chapter is data produced during the synthetic testing of the chess and analysis framework. The synthetic testing, as described in section 3.3.2, was concerned with the observation of various chess match scenarios as one was interested in uncovering the coherence between the properties of K1.

Furthermore as one was not certain how variation in performance would impact the various chess matches, the analysis chapter will look at data produced under specific conditions. The analysis will furthermore look at the outcomes of preliminary matches under various evaluation times, imposed penalties. This will is important in order to observe the key properties(K1) under various circumstances. The synthetic testing has been conducted with all of the matches being subject to varying conditions, in which a penalty has been imposed on the chess bots, or players, in order to mimic a certain degree of noise during the testing.

7.1 Synthetic testing

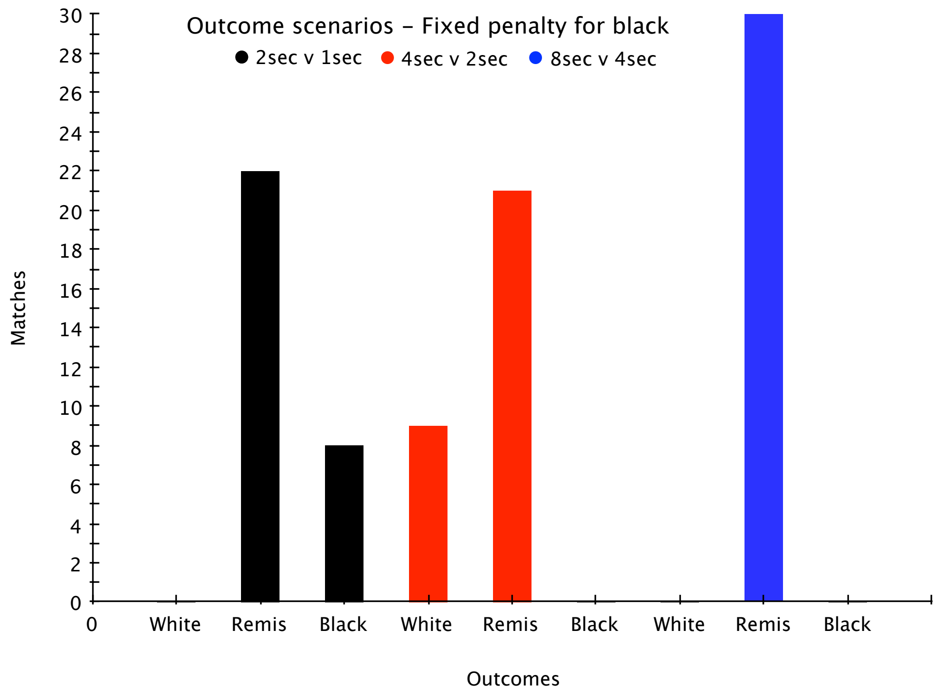


Figure 7.1: Graph displaying the outcomes of matches run with a fixed penalty.

Figure 7.1 illustrates the outcomes of three different 30 match sets that was run with a fixed penalty and where the evaluation time of Stockfish was set to the following

- White 2 seconds vs Black 1 second
- White 4 seconds vs Black 2 seconds
- White 8 seconds vs Black 4 seconds

Of the graph, in the 2vs1 second secnario, one may see that when running the matches where the black player has been given a penalty of 1 second, black comes out victorious in 8 matches, whereas white does not win a single match. The rest of the chess matches end in a draw. The following matches where the evaluation times was increased to that of 4 seconds and black was given a fixed penalty of 2 seconds, one may see that white wins a total of 9 matches whereas black doe not come out victorious in one match. Most of the matches still ended in a remis. The last set of bars represents the outcome where the evaluation time was originally set to 8 seconds, and where black was given a 4 second penalty, here one may observe that the matches all ended in remis, and neither player won a single match.

The outcome of the first batch of matches running with an evaluation time of 2 seconds was unexpected as the black player was given a penalty of 1 second, and would ultimately evaluate for a shorter duration, but still won 8 of the matches played. It is reasonable to assume that the reason for this is that the 1 second gap between the two players is not enough for Stockfish to do a significantly better evaluation. As when reaching a certain depth, 1 second will not impact the evaluation to the extend where Stockfish is able to find a better move.

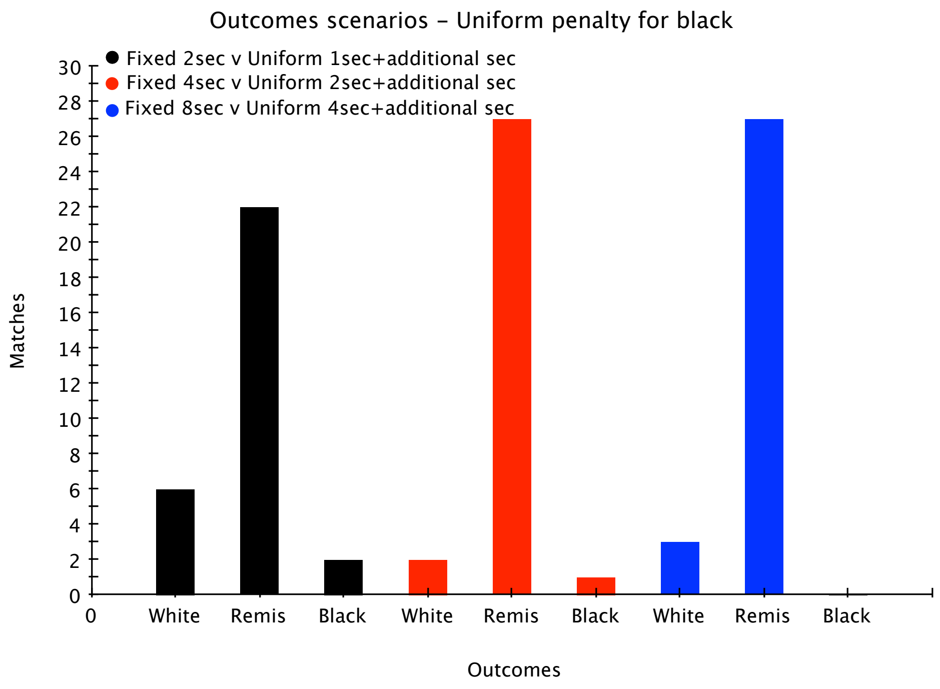


Figure 7.2: The graph displays the outcomes of matches where a uniform penalty has been imposed on the black player

Figure 7.2 represents the outcomes of chess matches where the black player was given a uniform penalty(see section 5.3.1). As with the set of matches described in the last section, the white player would run at the intended evaluation time of 2, 4 and 8 seconds. The black player had a uniform penalty imposed on its evaluation in which would make the evaluation time for the black player to fluctuate between the lower bound and upper bound of the evaluation time. This uniform penalty was imposed to mimic a more real scenario in where the chess bot experiences varying degrees of resources utilization. Of the graph one may observe that both the white and the black player comes out victorious in some of the matches in all but one scenario, although the number of draws are still high.

If one further observes the graph, one may see that as the evaluation time increases from 2 to 8 seconds the number of wins shared between the two players decrease, and that the number of draws increase. The reason behind this is that as the evaluation time increases, the further the chess engine will search, and at a certain depth the chess engine will spend more time evaluating as the number of moves needed to evaluate increases. And seeing as the black player has a fluctuating penalty between 4 and 8 seconds, the white and the black player may end up running the same evaluation time, in which may result in the two ending up with finding equally good moves.

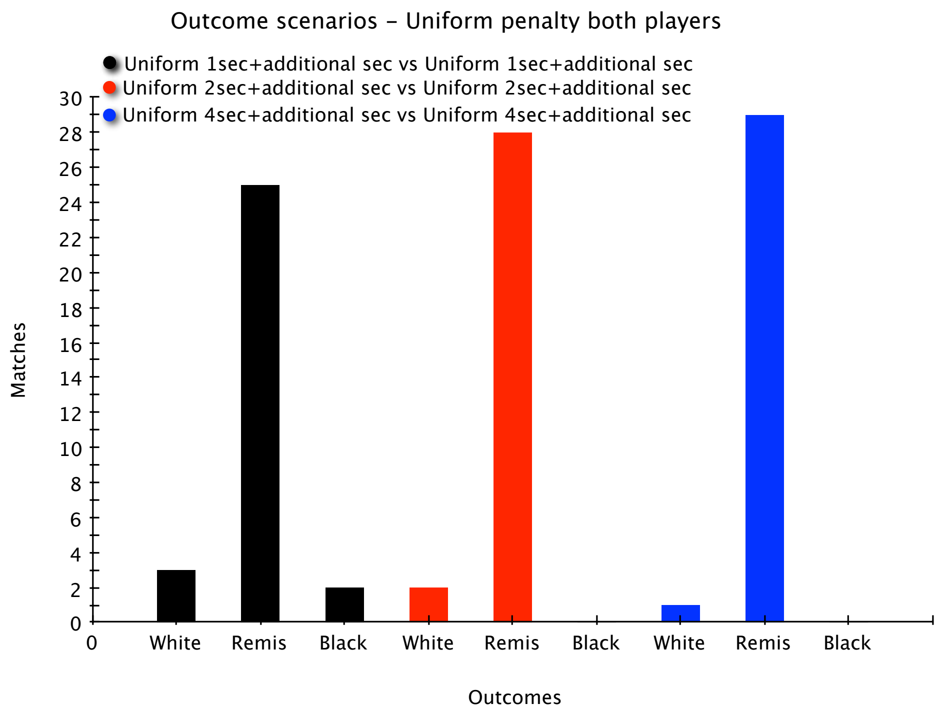


Figure 7.3: The graph above displays the outcomes where both players were given a uniform penalty

Figure 7.3 displays a graph that portrays the outcomes of chess matches having been played with uniform noise imposed on both players. Meaning that the evaluation time has been fluctuating for both parties. Of the graph one may observe that the outcomes are fairly similar to the ones displayed in graph 7.2. As one may see, the number of draws are quite substantial in contrast to that of victories for either player. Compared to the graph in the last section, this set of matches experienced an even lower number of wins, and higher rate of remis. This may be seen as a result of both parties now having the evaluation time running at various random intervals, within the fixed lower and upper bound of 2-1, 4-2 and 8-4 seconds. As a result, the two players may end up with quite similar evaluation times, which means that the two players may end up finding equally good moves.

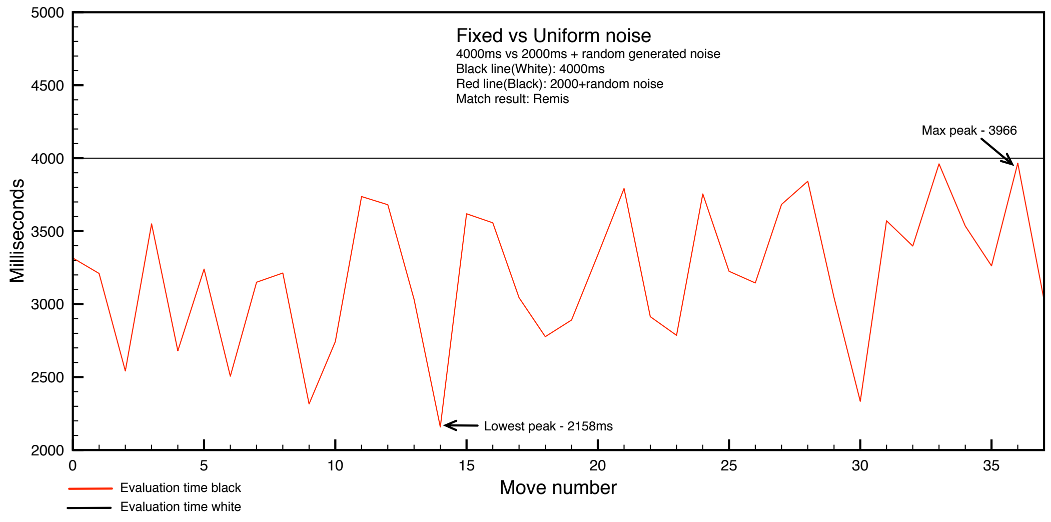


Figure 7.4: Graph showing how the uniform noise inflicts a chess bots evaluation.

One may observe of figure 7.4 how the uniform penalty imposed on the black player impacts the time in which he is to evaluate. One may observe that the white player evaluates at its given 4 second run time, and that the black player is significantly disrupted when initiating its evaluation. The uniform penalty imposed on the black player initially gives him a 4 second fixed penalty, which is equal to half the intended evaluation time, and then another random set of seconds are added to this penalty, having the lower bound being 2 seconds and the upper bound being the total number of seconds in which the players are allowed to evaluate, which is 8 seconds.

One may further observe that the black player experiences a distinct variation in evaluation time, and having its lowest peak at 2158 ms, and accordingly the highest at 3966 ms. This variation in evaluation is an interesting observation and gives the wanted indication towards how the engine is affected under varying conditions. This variation will also impact the number of nodes traversed by the engine.

,

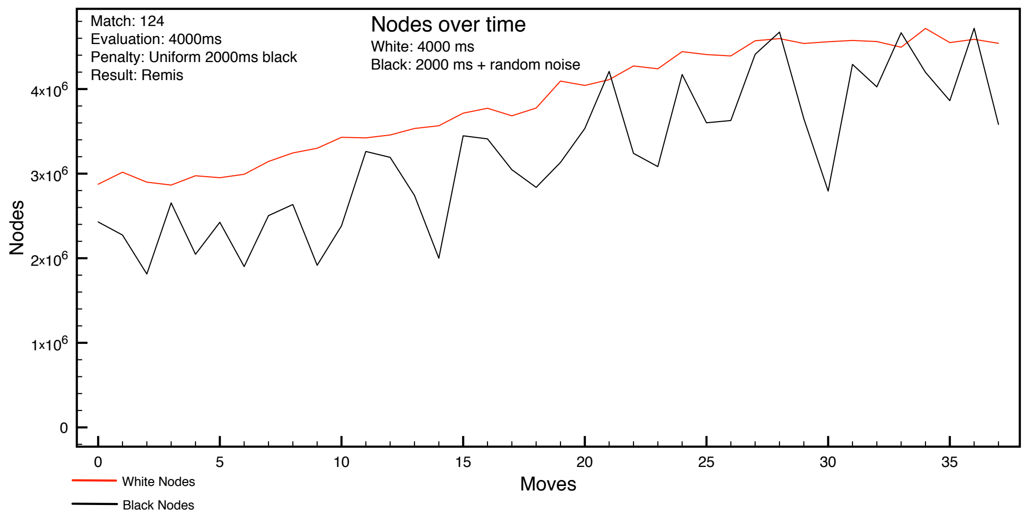


Figure 7.5: Illustrating how number of nodes traversed running with fixed and uniform penalty.

As mentioned in the latter part of the last paragraph figure 7.5 illustrates how the variation in evaluation time affects the overall nodes traversed by the chess engine. One can observe the white player, having a fixed evaluation time of 4000ms(4 seconds), has a stable node traversal throughout the represented moves, compared to that of the black player which has a fluctuating in evaluation time.

If one observers the black line throughout the graph, one can see how the number of nodes traversed by the black player fluctuates throughout the game. This fluctuation indicates that the engine is not capable of traversing the same amount of nodes as what would have been optimal, meaning that an impact on the performance of the engine affects the overall traversal of nodes.

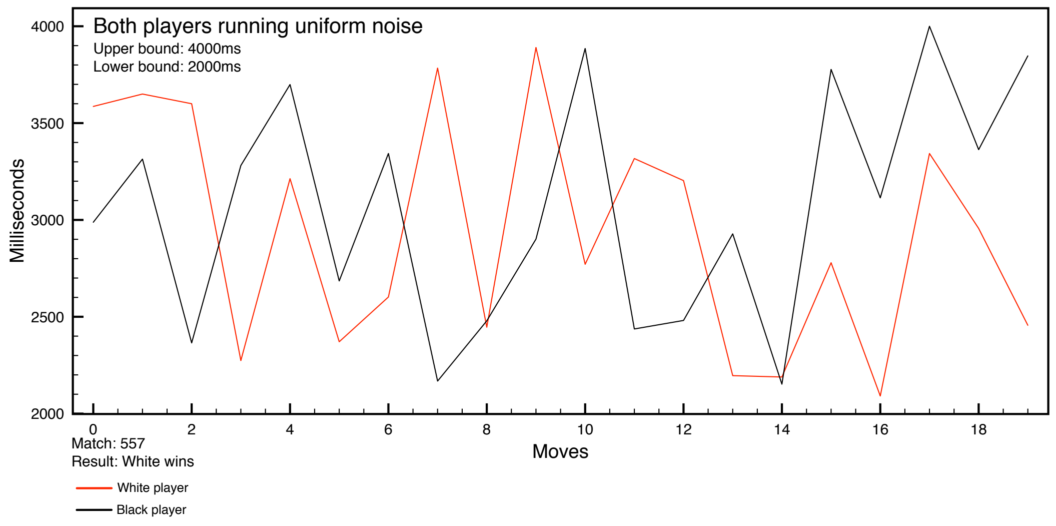


Figure 7.6: Graph showing how the uniform noise inflicts both chess bots during a match

If one views the graph depicted in figure 7.6, one may observe the scenario in which both players are subjected to noise. Both the white and black player have been running chess matches with an impeding uniform penalty. As described in figure 7.4, one may observe that both of the players are now experiencing variation in evaluation time when a match is undergoing. The two players experiences fluctuations between the lower and upper bound of 2 to 4 seconds. One may observe that the noise has an impact on the performance of the engine.

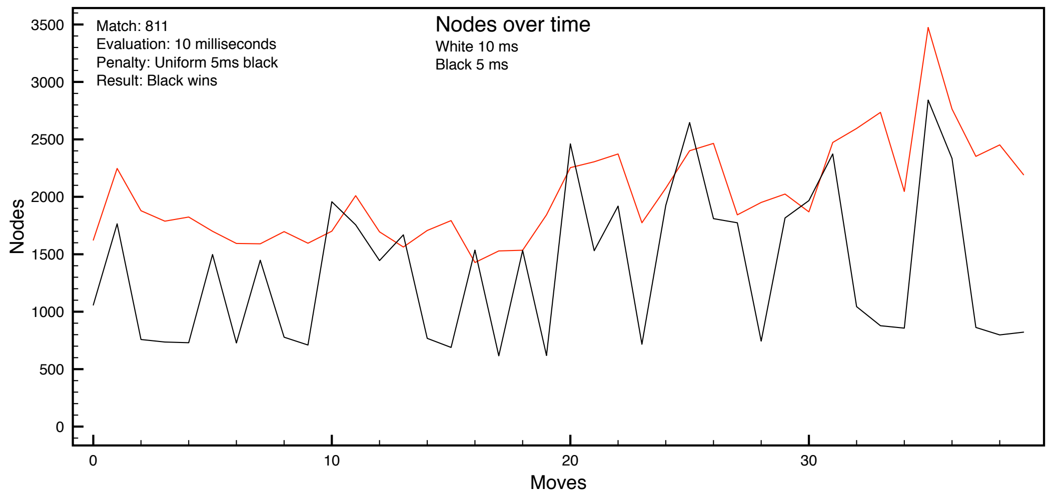


Figure 7.7: Illustrating how number of nodes traversed during extremely short evaluation time.

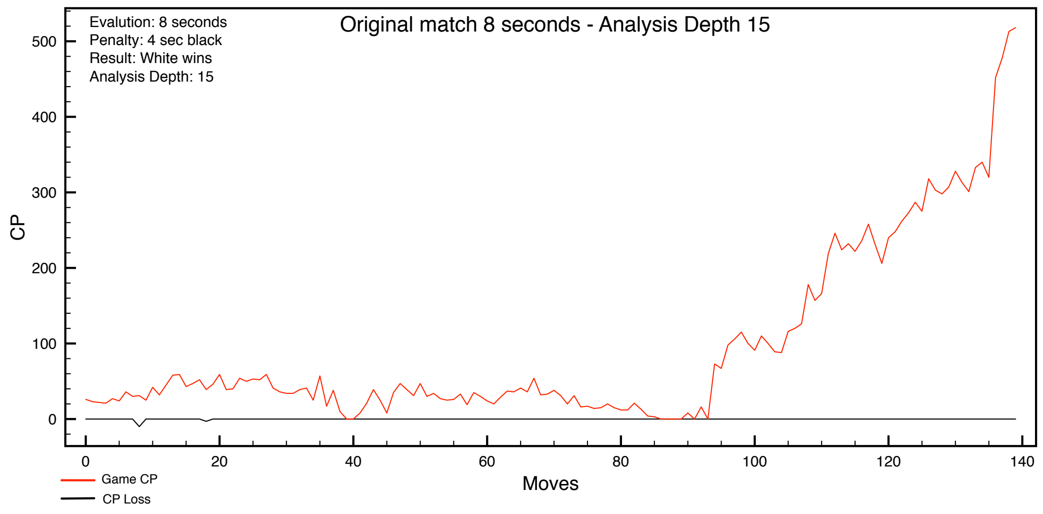


Figure 7.8: Analysis of a chess match running at 8 seconds vs 4 seconds evaluation time.

The graph illustrated in figure 7.8 represents the analysis of a chess match played out by two chess bots. This particular graph is concerned with the analysis of a chess match that has been run with an evaluation time of 8 seconds for white, and a 4 second penalty for the black player. Meaning that the black player would evaluate for half the original duration. The red line within the graph represents the changes in the centi pawn value as the original game progresses. What one may observe of the graph is the expected fluctuation in the centi pawn value, represented by *game_cp*, as the game is progressing. This due to white and black making moves that either increase of lower their advantage.

The black line relates to *CP_loss*, and as mentioned during 6.3, if one observes that the move made during the original game is also the move found to be best move by the referee during the analysis, one would expect to find the cp_loss value to be 0. As the moves found are the same. If the move found during the original match by the player does not correspond to that of the referees analysis, one would expect the move of the player to be of poorer quality, and the centi pawn loss would be represented by the black line displayed in the graph.

Observing the said graph one may furthermore see that the line representing cp_loss is quite flat, which one might expect as an evaluation time of 8 seconds is quite long, and the depth reached during the evaluation would be quite similar to that of the fixed depth set in the analysis. Meaning that the outcome of the evaluation performed during the original game by the players, compared to those done by the referee during the analysis were quite similar. This may be further verified as there are no significant spikes along the line represented by cp_loss.

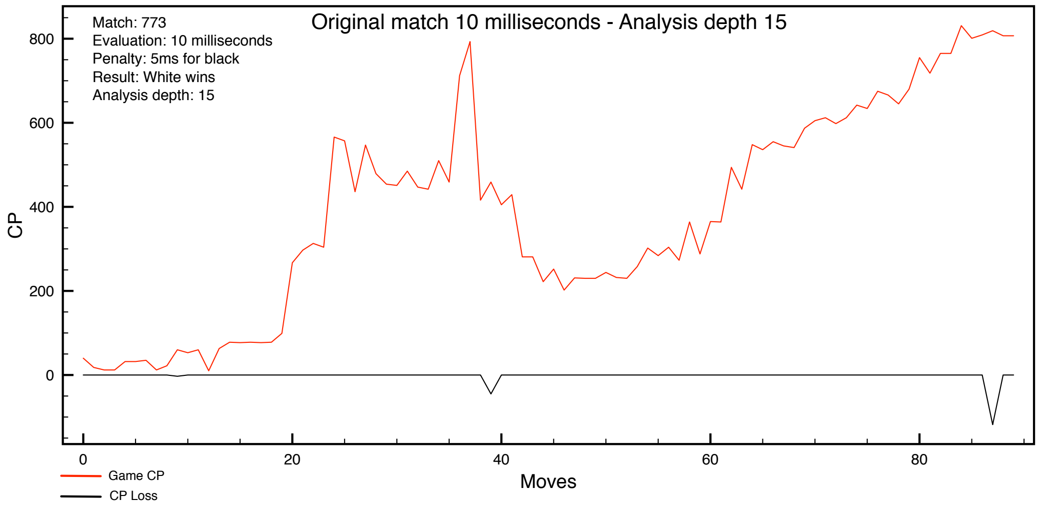


Figure 7.9: Analysis of a chess match running at 10 ms vs 5 ms evaluation time.

Figure 7.9 represents the analysis of a chess match in which the chess bots, more specifically the white player, ran with an evaluation time of 10 milliseconds and the black player with 5 milliseconds. The black player having a 5 millisecond penalty. As with the graph displayed in figure 7.8, the red line represents the changes in the centi pawn value throughout the match. Here one may also observes the fluctuation in the centi pawn value as the match progresses.

Compared to the analysis result in which one may observe in 7.8, where one expected to observe few variations in the cp_loss value as the evaluation time for the chess bots were quite long, one would expect for the current scenario to see quite elaborate differences in the cp_loss value. The reasoning behind this is that one would expect that the chess bots evaluations, in such an extreme short time, would find moves that are substantially poorer than to that of the referee bots during a far deeper analysis at a depth of 15.

By observing the graph one may see that this is in fact not the outcome. The line representing cp_loss does not show the expected spikes in terms of the cp_loss value fluctuating because of poor quality moves being made by the player, as would be the case if the player had come of with a move that was seemingly lower quality to that of the move evaluated by the referee in the analysis.

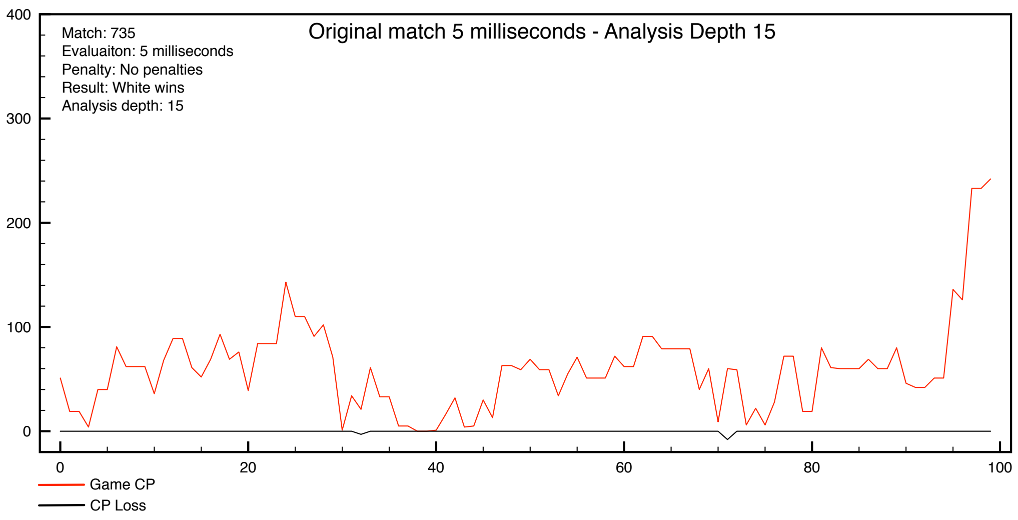


Figure 7.10: Analysis of a chess match running at 5 ms vs 5 ms evaluation time.

In figure 7.10 one may observe the graph in which the evaluation time of the chess bots have been lowered to that of 5 milliseconds, without any form of penalties. As with the scenario described above, one would have expected to see a change in the cp_loss value as the chess bots have been running their evaluation at such a short time span. One would have expected to see greater fluctuations in the cp_loss value, as the referees are evaluating at a deeper depth than the chess bots, and one would expect the moves made the players be poorer than those made by the referee.

Chapter 8

Discussion

The following chapter will discuss the conducted research, and aspects concerning the thesis. It will furthermore discuss alternatives to the approach in some specific areas, future work, and implementation of the framework and the related architecture. And furthermore ways to use the frameworks outside this research.

8.1 May this be conducted in another cloud environment?

One may easily conduct this research within another cloud environment, such as Amazon, although there are a few prerequisites:

- One is in need of the chess bot images
- The framework scripts
- A MySQL database

The chess bot images are created from snapshots of the original virtual machines in which portrays the chess bots. These images are running a Linux/Unix operating system and has Stockfish readily compiled on them, as well as the needed Xinetd service, in which is necessary in order to establish communication between the chess bots, and furthermore the referees. These images may be downloaded and uploaded to any cloud environment, such as Amazon.

Furthermore one may either develop some own form of framework in order to run similar scenarios concerning chess matches, or one may move the frameworks themselves, as they provide all the needed functionality in order to run chess matches and analysis, and moreover communicate with the chess bots, referees and the database.

Lastly one is in the need of storage medium, and the currently developed frameworks supports MySQL, so this needs to be installed on a separate virtual machine within the environment in which the frameworks are set to run. The frameworks contain the needed functionality in order to connect and communicate with the database, and one need only change the database specific information in order to connect.

8.2 Do the frameworks work?

This process has seen the creation of two individual frameworks, the chess framework and the analysis framework. The two frameworks were created each for its own purpose. The chess framework aimed at orchestrating a large array of chess matches in the cloud, and the analysis framework was to analyze these chess matches.

The Chess framework As mentioned in section 4, both of the frameworks would rely on automation as it would be a tedious task to manually arrange the vast number of chess matches and rounds of analysis intended during the research. The automation of the chess framework has been successful in the sense that it has been possible to initiate the frameworks to play a set of matches and subsequently analyze them.

The concept of batches as a way of organizing chess matches has proven to be successful, as one has been able to separate chess matches that has been run under various conditions in terms of penalties into their on respective chess batches. The integration of this solution has given one the opportunity of playing multiple matches simultaneously. Being able to do this greatly increases the efficiency of the framework, and as the framework has been automated, this benefits the overall orchestration of chess games. As one is, after the creation of a specific batch, in the position to initiate the framework and it will handle the matches without any interaction until all of the matches are finished playing.

The communication between the chess bots, channeled through the chess broker, has worked as intended in that it has been possible to run chess matches across the network between individual chess bots. The chess bots have been successful in playing out the matches as described by the chess batch they belong to, and the players have been successfully penalized during their matches. If any penalty was given. Furthermore has the chess framework, through the broker, been able to store the wanted chess game related data in the database for easy extraction when necessary.

The Analysis framework As with the chess framework the analysis was subjected to the concept of batches, although they were analysis batches in which had different specifications in order to meet the requirement of the analysis. The concept of analysis batches has proven successful as it has enabled the separation of analyzed data. This separation was necessary as extracting the data for inspection would be easier when having the various analyzed data grouped together in terms of the parameters they were subjected to during the analysis. Furthermore the automation of the analysis has proven effective, as an analysis that is initiated will be handled by the broker and the framework until its duration is over.

The referee bots are successful in handling their analysis in that they are able to fetch the correct data from the database through the communication of the broker. There is, however, a minor drawback in the analysis framework, in that when the referees are analyzing, several referees may analyze data from the same analysis batch. And because of this a mechanism was implemented in order to circumvent race conditions when multiple referees were fetching data from the database. The implemented functionality as of now locks the table rows within the database whenever a read operation has been initiated on the specific table, this may cause a small, not too significant, performance loss in the analysis process. The ramification of this locking and a proposed solution is further discussed in section 8.4. On the contrary, implementing queues adds an extra implementation task, and may be unnecessary and avoided as the current solution provides the functionality to circumvent race conditions.

8.3 Live Testing

As mentioned during the approach(section 3) the live testing would only become a reality if time permitted. The frameworks were to be moved into a live environment and the synthetic testing would act as a blueprint for the live testing.

The scenario under which the live testing would be conducted is different from that of the synthetic testing. During the live testing the chess matches would not be affected by penalties, meaning the matches will not be initiated with various penalties imposed of the various players. One does not want to impose penalties on the chess framework while the matches are running in the live environment as this would return misleading results, and one is interested in having the chess bots playing chess matches in what would be a real environment. Instead of such penalties, one would want to affect external factors within the environment that ultimately would affect the chess bots during their chess matches.

There are several scenarios under which the live testing could be conducted, as seen in figure 8.1. It is desirable that the chess broker, referee bots and the database is isolated from the environment in which the tests are conducted as they are not to be subjected to unwanted noise. Using processor affinity, or CPU affinity, you may lock, or bind specific processes to certain cores. One may consider the scenario where one have the chess bots running in an enclosed environment running on four cores(1), having each chess bot locked to one core each. One would then initiate the chess matches running in this scenario, and observe the outcome. This enclosing of chess bots and cores, provides opportunities for further experimentation with various forms of run time environments for the chess bots. As a way of mapping possible contention performance variation within the environment one may run experiments where one locks two chess bots to a single core in order to observe how they are affected during the game(2). One could then furthermore move on and place four chess bots on one core(3), and observe how the chess bots are impacted.

Moreover it may be possible to separate the chess bots, in the sense that two chess bots are isolated from the enclosed environment where each chess bot runs on their separate core. Whereas the chess bots inside the enclosed environment are locked to the same core(4). One may then look at the outcomes and link the mapping of contention between the two scenarios. All of these scenarios may be used in order to fully map the possible contention between the chess bots in cloud environment.

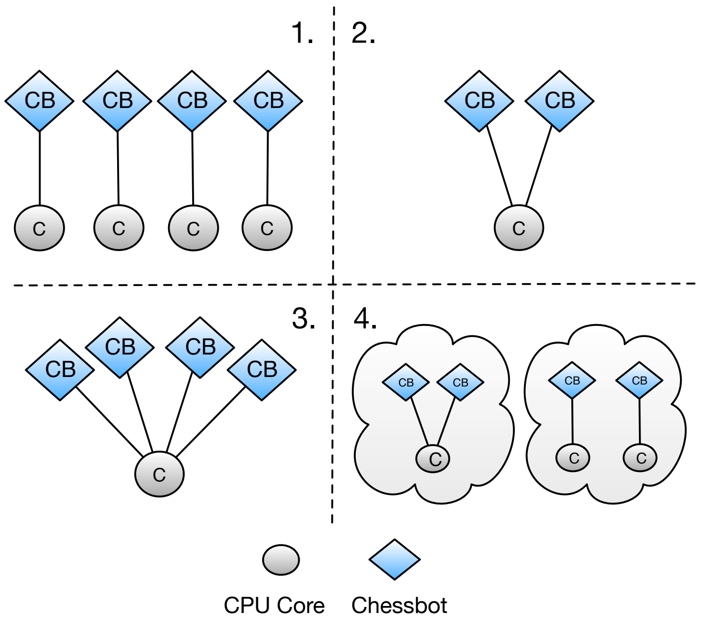


Figure 8.1: The above illustration depicts the various scenarios just described

8.4 Implementation of queues - Considering RabbitMQ

The analysis framework utilizes a functionality in order to prevent race conditions when several referee bots are analyzing the same analysis batch. These race conditions may be bad for the overall performance of the framework as it will allow analysis of the same data more than once, which is a waste of resources and it will generate duplicates within the database. As described during section 6.8 the analysis framework has been developed with a mechanism for preventing unwanted race conditions. As of the end of this research, the analysis framework utilizes a built in function in the MySQL database in order to lock specific rows for reading when a read operation on a given row is on going.

This mechanism provides the needed functionality in that it allows for better handling of read operations in terms of having simultaneous requests to the database any given second. There is an imminent drawback in that there is sort of a "first come, first serve" principle in that when the chess broker is running several referees simultaneously and they need to access the database all at the same time, the broker will allow the first referee to check for the latest move to analyze, but will deny all of the other referees access. This will impede on the performance of the analysis, as it will delay and create unnecessary gaps in the analysis process. Even though not too significant, it is a factor to consider.

A solution, and a different approach that may improve on this impeding factor is that of implementing a queuing service, e.g RabbitMQ. The reason this might considerably enhance the analysis is that instead of having the database locking a row in the database each time a referee wants to perform read operation, all of the referees may collect moves for analysis from its own queue. This would be implemented in the way that in addition to having the database isolated on its own VM, one could implement RabbitMQ on a separate virtual machine. On could also have implemented it on its own physical server, as this might even further enhance the capacity of RabbitMQ. Although it depends of the activity towards the database in terms of where one may consider implementing the RabbitMQ service.

Each of the referees would have been assigned to their own queue, and the move data would be sent to the RabbitMQ handler, and then the respective referees would then consume the data from the queues that they have been assigned to.

8.5 Does the analysis work?

The conducted research had a goal of successfully mapping success in a game of chess to that of resource contention. As described throughout the process, specific success criteria were chosen in order to map the cohesion between the two. One of which was the centi pawn value, the other nodes over time. The centi pawn value was chosen as a success criteria as it is portrays the current advantage for any of the two players during a game. The goal was to run a set of chess matches, and then analyze the moves produced from these matches, with the focus on the potential loss in quality of the moves made by the players compared to what the referees found during the analysis. Based on the success rate of the individual moves.

Meaning that one expected to observe a significant loss in quality in the moves made by the player if the moves made were different to what the referees found on a considerably deeper evaluation depth. One would expect to see a fluctuation in the centi pawn value represented by cp_loss, as this would indicate that the moves found showed a centi pawn value in which would be greater than if the moves made my the player and the referees were identical.

As described during the analysis chapter 7 the expectations towards the findings in graph 7.5 were that the cp_loss value would stay close to 0 throughout the game as the evaluation time of the players were quite long, and because of that they would have evaluated to a depth very similar to that of the referee bots. Because of this the majority of moves made during the game would end up identical. So the quite flat line observed in the graph was expected.

As the evaluation time of the players lowered considerably, one expected a change in the cp_loss value as the players were now running significantly lower evaluation times and would accordingly end up much lower evaluation depths. Although if one observes the graphs 7.9 and 7.10 one may see that there are no significant fluctuations in the cp_loss value. Which is unexpected as one would have expected that the referees would have come up with considerably better moves at the depth at which was analyzed compared to what was run by the players.

This unexpected behavior may be caused by the input to the framework, or the analysis itself and may be the reason behind the inconclusive result of the analysis of specific matches. One may argue that it perhaps is the processing of the data during the analysis that is not efficient enough. Meaning that the approach taken in calculating the loss in quality of the moves are not correctly adjusted.

8.5.1 Better understanding the analysis

In retrospect, there might be ways to try to better understand the analysis and how the values behave, and to more correctly calculate the cp_loss value for a given move. The chess bots, or players, are running their evaluation based on time, and as of now one is not observing the expected outcome of the analysis. It might be possible to rig the cp_loss value better if one instead of having the players running the evaluation on time, dictate the depth at which they are to evaluate.

Although controlling the depth of the evaluation contradicts the purpose of the research and is not the same type of "sabotage" as one would have with inflicting penalties on the evaluation time. This is because one would have to make rough assumptions that a given evaluation time would lead to a specific depth. And one would rig the game just to confirm assumptions made as to how the outcome would have been for a certain evaluation time, and furthermore in order to observe if one obtains the wanted result. Although this is not in line with current research it might have been used as a method to better understand the evaluation and the values used to measure success, in order to more adequately calculate the cp_loss value.

8.6 What can system administrators learn from this?

In the future for the branches within the industry that host gaming services, resource locking is something that may become a challenge, not only for the industry itself, but for the system administrators that are responsible for the administrating the game servers. There are services being hosted that achieve success, for instance Netflix and Dropbox, are hosted in Amazon. But these services are asynchronous, for instance, Netflix is not an immediate service, as Netflix will buffer the content before streaming it to the user, and one can not buffer a game.

And with this type of sensitivity in terms of seamless hosting one may envision that the cloud as it is now, may not have the same prerequisites for game hosting as it does streaming, but one may consider the scenario where one may host game servers on more prioritized machines, in terms of locking the resources to specific machines. It would be possible with the technology of today, but it might force an evolution in terms of hosting game servers. As the services hosted today do not have the same issues as a game service.

That being said, although we witnessed a slight variation in terms of performance when playing chess, other games may be harder to measure. In the sense that it would not have been possible to create an equivalent framework as the one created in this thesis, in relation to a first person shooter. How could one go about observing and verify how the penalty inflicts another type of game? As long as one has the possibility of running a simulation of the given game, it is possible to recreate a similar scenario, the essence of the framework created will still be valid in the sense that one could still organize matches played with batches, but one would probably not be concerned with individual moves as one is in chess like scenario. One would still have the possibility of graphing the outcomes of matches as has been done during this research.

8.6.1 Game success monitoring

If the system administrator has the possibility of extracting the success rate related to the played matches, in terms of who is winning and losing, it is may be possible to create an expectation of success. A system administrator usually does not have access to "game play logic", meaning that the system administrator has no way of knowing how well a specific opponent or server is performing or how successful it is.

Although if the system administrator has access to information in terms of how successful the server is, he may to a greater extent, observe how the variation in success propagates. The system administrator can use this knowledge in sense that, if an opponent is initiated with a difficulty level of "hard" it is expected that this opponent adheres to this level. If the this opponent however, is losing frequently, one may have an issue in terms of resources. In the sense that he is under achieving in relation to the expected level.

8.7 Future Work

One could have further developed more tools working in conjunction with the frameworks in order to retrieve summaries from the specific batches, in terms of wins, losses and remis. Moreover one could have moved the frameworks to another cloud environment after conducting needed tests in the current environment. This is order to look at how these environments impact the matches run. For instance one could have moved the frameworks to Googles Amazon cloud environment.

In terms of the analysis framework designed and implemented in this thesis, there are a few aspects to consider when looking at further development. As discussed in section 8.4, the analysis framework implemented as of right now may be further improved by implementing a form of queuing service. This may ease the job of the broker, and furthermore help prevent the race conditions that are likely to occur, without the impeding effect that the current solution impose.

Furthermore as discussed in section 8.5.1, one may consider during future development, running more controlled preliminary tests with the main focus being on running the evaluation on a specific depth in order to better clarify the behavior of the variables used as a measure for success in order to more adequately calculate the loss in quality of move. This means obtaining a clearer understanding of the values related to success.

Chapter 9

Conclusion

The thesis has seen exploration through the successful creation of a set of frameworks meant for running and analyzing games. The work put down is this research may easily be continued by utilizing and tweaking these frameworks. The frameworks that have been created are already running in the cloud, and may be moved to other cloud environments as long as the environment fulfills certain criteria.

Concerning the investigation conducted one may look at the results in the sense that if one is given less time to evaluate, or think, the likelihood of winning is considerably reduced. A reduction in half a second considerably affects the likelihood of winning. This confirms our assumptions that game types that depends on quite substantial computation will suffer in an environment where resources vary to a certain degree. Although, for game types that has considerably longer periods of evaluation, there is no substantial evidence that support these assumptions. This may concern turn based and strategy based games. One may then generalize this and say that the shorter the evaluation period, the more vulnerable it is. Meaning that the success rate is reduced.

Furthermore one wanted to go deeper and investigate why this is the case, and for chess in general, one wanted to investigate the quality of individual moves made during a game of chess. This has seen partly successful results. One received some unexpected results and it is clear that more work needs to be conducted and further investigation is needed. Nonetheless, the results were promising, but difficult to interpret. In order to further investigate this and obtaining a better understanding one may look to the method described in section 8.5.1, concerning using depth during preliminary testing.

As stated during the introduction of this thesis, one was introduced to the scenario in which one was playing a game against an opponent. Envision that you are now playing against this opponent, and try and imagine how this would play out if you had half a second less to evaluate any given situation. This research has shown that if this game had run on a machine in the cloud, that varying degrees of resources and contention is something that one should be wary of.

Bibliography

[1] Cegt rating list. http://www.husvankempen.de/nunn/40_4_Ratinglist/40_4_BestVersion/rangliste.html. Last visited March 9th 2014.

[2] Chess engine grand tournament. http://www.husvankempen.de/nunn/. Last visited March 9th 2014.

[3] Computer chess rating list. http://www.computerchess.org.uk/ccrl/4040/index.html. Last visited March 9th 2014.

[4] Gnu general public license. https://www.gnu.org/copyleft/gpl.html. Last visited March 9th 2014.

[5] Komodo home page. http://komodochess.com/. Last visited March 9th 2014.

[6] Simen Agdestein. *Simens Sjakkbok*. Gyldendal Norsk Forlag ASA, 1997. ISBN: 82-05-25146-0.

[7] Wikimedia Beao. Algebraic notation representation. http://en.wikipedia.org/wiki/Algebraic_notation_(chess). Last visited March 8th 2014.

[8] Wikimedia Beao. Algebraic notation representation. http://commons.wikimedia.org/wiki/File:SCD_algebraic_notation.png. Last visited March 8th 2014.

[9] Wikimedia Beao. Algebraic notation representation. http://commons.wikimedia.org/wiki/Commons:GNU_Free_Documentation_License_1.2. Last visited March 8th 2014.

[10] Sara Bouchenak, Gregory Chockler, Hana Chockler, Gabriela Gheorghe, Nuno Santos, and Alexander Shraer. Verifying cloud services present and future. http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/40816.pdf. Last visited March 12th 2014.

[11] Programmed by Martin Blume and GUI Website by Wilhelm Hudetz. Arena chess gui. http://www.playwitharena.com/. Last visited March 10th 2014.

[12] Chessprogramming.com. Centipawns. http://chessprogramming.wikispaces.com/Centipawns. Last visited March 9th 2014.

[13] Chessprogramming.com. Chess engine evaluation function. http://chessprogramming.wikispaces.com/Evaluation. Last visited March 9th 2014.

[14] Chessprogramming.com. Chess material. http://chessprogramming.wikispaces.com/Material. Last visited March 9th 2014.

[15] Chessprogramming.com. Chess piece relative value. https://chessprogramming.wikispaces.com/Point+Value. Last visited March 9th 2014.

[16] Amazon EC2. Amazon elastic cloud. http://aws.amazon.com/ec2/. Last visited March 12th 2014.

[17] Cloud Gaming. Cloud gaming report 2012 distribution and monetization strategies to increase revenues from cloud gaming. http://www.cgconfusa.com/report/http://www.cgconfusa.com/report/documents/Content-5minCloudGamingReportHighlights.pdf, 2012. Last visited 26nd of February, 2014.

[18] Google. Google app engine. https://developers.google.com/appengine/docs/whatisgoogleappengine. Last visited March 11th 2014.

[19] Rackspace Hosting and NASA. Openstack open source cloud software. https://www.openstack.org/. Last visited March 25th 2014.

[20] Robert Houdart. Houdini home page. http://www.cruxis.com/chess/houdini.htm. Last visited March 9th 2014.

[21] IBM. Ibm and hp virtualization a comparative study of unix virtualization on both platforms. https://www.ibm.com/developerworks/aix/library/au-aixhpvirtualization/au-aixhpvirtualization-pdf.pdf. Last visited March 3rd 2014.

[22] Fédération internationale des échecs(FIDE) or World Chess Foundation. Fide handbook(laws of chess) under appendix c: Algebraic notation. http://www.fide.com/component/handbook/?id=125&view=article. Last visited 10th of February 2014.

[23] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On the performance variability of production cloud services. http://www.pds.ewi.tudelft.nl/~iosup/tech_rep/cloud-perf-var10tr.pdf/. Last visited March 12th 2014.

[24] Stefan-Meyer Kahlen. Universal chess interface(uci). http://wbec-ridderkerk.nl/html/UCIProtocol.html. Last visited March 10th 2014.

[25] Stefan-Meyer Kahlen and Rudolf Huber. Universal chess interface(uci). http://www.shredderchess.com/chess-info/features/uci-universal-chess-interface.html. Last visited March 10th 2014.

[26] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. http://ieeexplore.ieee.org.ezproxy.hioa.no/stamp/stamp.jsp?tp=&arnumber=4211036. Last visited 27th of January, 2014.

[27] Jaideep Moses, Ravi Iyer, Ramesh Illikkal, Sadagopan Srinivasan, and Konstantinos Aisopos. Shared resource monitoring and throughput optimization in cloud-computing datacenters. http://ieeexplore.ieee.org.ezproxy.hioa.no/stamp/stamp.jsp?tp=&arnumber=6012910. Last visited March 11th 2014.

[28] Tord Romstad. Glaurung chess. https://chessprogramming.wikispaces.com/Glaurung. Last visited March 9th 2014.

[29] Jorg Schad, Jens Dittrich, and Jorge-Arnulfo Quiane-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. http://dl.acm.org/citation.cfm?id=1920902&dl=ACM&coll=DL&CFID=301673206&CFTOKEN=59311921. Last visited March 11th 2014.

[30] Claude E. Shannon. Programming a computer for playing chess. http://vision.unipv.it/IA1/ProgrammingaComputerforPlayingChess.pdf. Last visited March 8th 2014.

[31] Marcio Silva, Kyung Dong Ryu, and Dilma Da Silva. Vm performance isolation to support qos in cloud. http://ieeexplore.ieee.org.ezproxy.hioa.no/stamp/stamp.jsp?tp=&arnumber=6270766. Last visited March 12th 2014.

[32] Daylen Yang. Stockfish home page. http://stockfishchess.org/. Last visited March 9th 2014.

[33] Zynga. Zynga. http://zynga.com//. Last visited March 5th 2014.

[34] Zynga. Zynga revenue. http://investor.zynga.com/releasedetail.cfm?ReleaseID=738074. Last visited March 5th 2014.

Chapter 10

Appendix

Chess framework script

Chess framework - chess.pl

```
#!/usr/bin/perl
use strict vars;
use Getopt::Std;
use IO::Socket;
use DBI;

#DATABASE VARIABLES#

#MySQL Server HOST IP ADDRESS
my $DBHOST = "*";
#MySQL Server PORT
my $DBPORT = "13306";
#The database to use when connecting to the server
my $DB     = "*";
#The user in which is used to login with
my $DBUSER = "*";
my $DBPASS = "*"; #Password of the current user

#SWITCHES NEEDED FOR THE SCRIPT#

my $opt_string = "vhB:b:d:D:e:i:m:M:n:p:r:t:w:x:P:";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

my $VERBOSE      = 1 if $opt{'v'};
my $DEBUG        = #1 if $opt{'d'};
#Switch for batch creation or listing
my $BATCH        = $opt{'B'};
#Switch for IP address of white player
my $WHITEIP      = $opt{'w'};
#Switch for IP address of black player
my $BLACKIP      = $opt{'b'};
#Switch to determine type of distribution
#(ex: Poisson, normal)
my $DISTRIBUTION = $opt{'D'};
#Switch providing maximum number of moves
#allowed during the matches of this batch
my $MAXMOVES     = $opt{'m'};
#Switch which names a given batch
my $BATCHNAME    = $opt{'n'};
#Switch enabling one to describe a batch
my $BATCHDESC    = $opt{'e'};
#Switch issued to start a chess match
my $PLAY         = $opt{'p'};
#Switch determining the maximum time each side
#is allowed to search for moves(stockfish)
my $MAXTIME      = $opt{'t'};
```

```
#Switch enabling one to choose which batch
#to list(All will be shown if not given)
my $BATCHID      = $opt{'i'};
#Specify a depth to run the chess engine
my $DEPTH        = $opt{'d'};
#Specify a movetime to run the chess engine
my $MOVETIME     = $opt{'M'};
#Counts number of matches
my $MATCHCOUNT   = $opt{'x'};
#Input a range for the random generator
my $randRange    = $opt{'r'};
my $PENALTY_STRING = $opt{'P'};


## TODO: Corresponding columns in batch table
my $WHITE_PENALTY_TYPE;
my $WHITE_PENALTY_DIST;
my $WHITE_PENALTY_RANGE;
my $WHITE_PENALTY_MEAN;
my $BLACK_PENALTY_TYPE;
my $BLACK_PENALTY_DIST;
my $BLACK_PENALTY_RANGE;
my $BLACK_PENALTY_MEAN;

if ( $opt{'h'} ){
    usage();
    exit 0;
}

#CONNECTING TO MYSQL DATABASE#

my $dbh = DBI->connect("DBI:mysql:$DB;host=$DBHOST /
;port=$DBPORT",$DBUSER,$DBPASS) or die $DBI::errstr;
if ( $dbh ){
    print "Connection to $DBHOST and $DB successfull\n";
}


#Checking if correct swithces are provided with the script
usage() and die "Need to supply correct /
switches \n" unless $BATCH or $PLAY;

#BATCH CREATION AND BATCH LISTING#
#Code for batch creation and batch listing

 if ( $BATCH && $BATCH =~ /create/ && $BATCHDESC && /
  $BATCHNAME && $DISTRIBUTION && /
  $PENALTY_STRING && $MAXMOVES && $MAXTIME  ) {
  print "New batch in the making\n";
## TODO: Insert values for penalties ( for black and white )
  parsePenaltyString();
  batch_creation($BATCHDESC,$BATCHNAME,$DISTRIBUTION, /
  $MAXMOVES,$MAXTIME,$WHITE_PENALTY_TYPE, /
  $WHITE_PENALTY_DIST,$WHITE_PENALTY_RANGE, /
  $WHITE_PENALTY_MEAN,$BLACK_PENALTY_TYPE,/
  $BLACK_PENALTY_DIST,$BLACK_PENALTY_RANGE /
   $BLACK_PENALTY_MEAN);
} elsif ( $BATCH && $BATCH =~ /list/ && $BATCHID ) {
  print "Now listing wanted batch(es)\n";
  batch_listing($BATCHID);
} elsif ( $BATCH =~ /list/) {
  print "Now listing wanted batch(es)\n";
  batch_listing();
}

##CODE THAT INITIATES A GAME OF CHESS##

#Preparing statements for adding information about /
```

```
players to batch.
my $matchSelect = $dbh->prepare /
("SELECT match_id FROM matches");

my $batchUpdate = $dbh->prepare /
("UPDATE batch SET ip_w='$WHITEIP', /
ip_b='$BLACKIP' WHERE batch_id = ?");

my $batchSelect = $dbh->prepare /
("SELECT * FROM batch");

#IP of the players(White and Black)
my $player = $WHITEIP;

#Proceding to start a chess match and /
#adding information of the two players in the batch table.

if ( $PLAY && $PLAY =~ /play/ && $BATCH /
&& $MATCHCOUNT && $WHITEIP && $BLACKIP ) {
print "Chess match commencing ###  /
White: $WHITEIP Vs black: $BLACKIP ###\n";
print "Updating batch with id: $BATCH, /
with player information unless already updated\n";

$batchSelect->execute();
  while (my @row = $batchSelect->fetchrow_array()) {
   for (my $i=0; $i<$#row; $i++) {
    if ( $row[0] == $BATCH && $row[4] == NULL && /
         $row[9] == NULL ) {
    # TODO:  Fetch penalty variables and
    #insert into global variables
    $batchUpdate->execute($BATCH);
        }
   }
 }

#MATCH INITIATED - MAIN SCRIPT STARTING - PLAY MATCH#

#MAIN SCRIPT VARIABLES#
my $chessboard  = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/
        RNBQKBNR w KQkq - 0 1"; #The initial match board.
#Newfen is set with the initial chessboard when
#starting a match.

my $newfen  = $chessboard;

#Set to ensure that the game is played in a loop. If true.
my $match_going = 1;

#If set to 1, a new match will commence, if 0 = false,
#a match is still in progress.

my $newMatch = 1;
#TEMP VARIABLE, WILL BE REPLACED BY
#DATA FROM MAX_TIME IN THE DB.

my $movetime;

#TEMPORARY - Ensures that only the given
#number of moves are played.

my $count = 0;

#ID of the current match being played,
#fetched for insertion of move data.
my $MATCHID;

#Sets the blackmove to 0 for the first
```

```
#move(Which is white).
my $blackmove   = 0;

#LoopCounter for adding HALF MOVES for @
#white and black player
my $loopCounter = 0;

#Denotes which move number is being made.
#Increases only when each have made one half move
my $move_number = 1;
my $matchcount = 0;
my $moveTimePen;

while ( $match_going ){
  if ( $newMatch == 1 ) {
  $move_number = 1;
  $player = $WHITEIP;
  $blackmove = 0;
   if ($matchcount == $MATCHCOUNT) { #Ends the current /
    batch if the number of matches reaches a /
    pre-set number(e.g "-x 10")
    exit 0;
}
my $HINT = int(rand(10000));

$dbh->do("INSERT INTO matches (batch_id,timestamp_start,
hint) VALUES($BATCH,NOW(),$HINT)");
#GETTING CURRENT MATCH ID

my $findMatch = $dbh->prepare("SELECT match_id FROM /
matches WHERE hint = $HINT");

$findMatch->execute();

my @row = $findMatch->fetchrow_array();
$MATCHID = $row[0];
$dbh->do("UPDATE matches SET hint=0 /
WHERE match_id = $MATCHID");
}
print "The current matchID is: $MATCHID\n";

#Running the run_match sub which starts the match
$newfen = run_match($player,$newfen);
print "got fen after one move: '$newfen'\n";

#Checks if it is White or blacks turn, also sets the /
blackmove variable.
if ( $player eq $WHITEIP ){
  $player = $BLACKIP;
  $blackmove = 1;
} else {
  $player = $WHITEIP;
  $blackmove = 0;
}

#Code for making sure the move number is the
#same for black and white, half moves.
 if ( $loopCounter %2 > 0) {
 $move_number++;
 print "CURR MOVE NUMBER: $move_number\n";
 }

# Checking that it does not SURPASS MAX MOVES!
#STOPS THE MATCH IF IT DOES
#$batchSelect->execute();
# while (my @row = $batchSelect->fetchrow_array()) {
#  if ( $row[0] == $BATCH && $count > $row[3] ){
#   $match_going = 0;
# }
#}
$count++;
$loopCounter++;
 }

#Sub routine that initiates the chess match between two /
virtual machines running an instane of stockfish.
sub run_match {
  my $opponents = $_[0]; #IP addresses of the black and /
  white
   player(Two virtual machines)
  my $fen = $_[1]; #The Chess board

#Connecting to a socket on virtual machines hosting
#a chess server.
my $socket = new IO::Socket::INET (
PeerAddr  => $opponents,
PeerPort  =>  3333,
Proto => 'tcp',
) or die "Couldn't connect to Server\n";

my $lastline = "";
if ( $socket ){

# 1. Send brett
print $socket "position fen $fen\n";

# 2. beregn trekk
$batchSelect->execute();
while ( my @row = $batchSelect->fetchrow_array() ) {
if ( $row[0] == $BATCH ) {
 if ( $MOVETIME ) {
 #$MOVETIME = $row[5];
 my $movetime = calculatePenalty($MOVETIME,$BATCH);
 print $socket "go movetime $movetime\n";
 print "THIS IS CURRENT MOVETIME: $movetime\n";
 $moveTimePen = $movetime;
}
elsif ( $DEPTH ) {
print "DEPTH $DEPTH\n";
print $socket "go depth $DEPTH";
print "THIS IS DEPTH: $DEPTH\n";
 }
}
}
my $bestmove; #The best possible move
while ( my $line = <$socket> ) {
chomp $line;
 #print "$line\n";
#Example of lastline: info depth 18 seldepth 26 score /
#cp 26 nodes 1863996
 $lastline = $line if ( $line =~ /info depth / );
 if ( $line =~ /bestmove (\S+) / ){
 $bestmove = $1;
 last;
 }
 }
print "$lastline\n";
print "running move $bestmove\n";
#print "$bestmove\n";

# 3. Gjennomfør trekk
print "BLACK MOVE IS: $blackmove\n";

#Calling Check_move function check wether mate or draw
my $check_move = check_move($player,$bestmove, /
$fen,$MATCHID);
```

```

#checking return values for what to do next, either start /
#a new match or continue playing
if ( $check_move == 1) {
 $newMatch = 1;
 move_data($lastline,$MATCHID,$move_number,$blackmove, /
 $fen,$bestmove,$moveTimePen);
 $matchcount++;
 print "MATCHCOUNT: $matchcount\n";
 $newfen = $chessboard; #The initial match board.
}elsif ( $check_move == 0 ) {
 print $socket "position fen $fen moves $bestmove\n";
 print "MAKING A NEW MOVE: $bestmove\n";
 $newMatch = 0;

#ADDING MOVE DATA TO THE "MOVES" table. See move_data sub.
move_data($lastline,$MATCHID,$move_number, /
$blackmove,$fen,$bestmove,$moveTimePen);
print "ADDING TO DB\n";
# 4. få ny fen
 print $socket "d\n";
 my $newfen;
 while ( my $line = <$socket> ) {
 print "D-DATA: $line";
 #info depth 18 seldepth 26 score cp 26 nodes 1863996
 if ( $line =~ /Fen: (.*)$/ ){
 $newfen = $1;
 chomp $newfen;
 }
 if ( $line =~ /Legal / ){
 last;
 }
}
return $newfen;
print "Conversation finished\n";
print "socket still active\n" if $socket;
close($socket);
  }
 }
}
}

#SUB ROUTINE THAT CHECKS FOR WIN OR DRAW

#The CUT FET is a FEN without the last 5 fields.
my $cut_fen;
my $fen_repeat = 0; #The number the same FENs have occurred.
sub check_move {
#Regex that matches only the FIRST field of the input FEN
 if ( $_[2] =~ /(.+\/\w+)/ ) {
 $cut_fen = '%'.$1.'%';
 }
#Fetching the input FEN from the DATABASE and putting /
#it in $fen_repeat.

my $fenFetch = $dbh->prepare("SELECT COUNT(*) as /
fen_count FROM moves WHERE /
match_id = '$_[3]' and fen LIKE '$cut_fen'");

$fenFetch->execute();
my @row = $fenFetch->fetchrow_array();

my $fen_repeat = $row[0];
print "FEN-REPEAT: $fen_repeat\n";

#Checks if the same FEN occurs 3 times within /
#the same match, if so it is a DRAW.

```

```
if ( $fen_repeat == 2 ) {
$dbh->do("UPDATE matches SET result='remis', /
timestamp_finished=NOW() WHERE match_id = $_[3]");
return 1;
print "THE GAME IS A DRAW\n";
}

#Checks if the position is CHECK MATE, as is the /
#case if a "NONE" move is returned.
elsif ( $_[1] =~ /\(none\)/ ) {

if ( $_[0] eq $WHITEIP ) {
$_[0] = $BLACKIP;
}else{ $_[0] = $WHITEIP }

#REMEMBER TO ADD TOTAL MOVES TO THE MATCHES TABLE
$dbh->do("UPDATE matches SET result='win_$_[0]', /

timestamp_finished=NOW() WHERE match_id = $_[3]");
print "ITS CHECK MATE PEOPLE\n";
return 1;
}
else {
return 0;
 }
}

#SUB ROUTINE THAT ADD MOVE DATA TO THE MOVES TABLE
my $nodes;
sub move_data {
print "Current pen: $_[6]\n";
my @moveData = split(' ', $_[0]);
 if ( $moveData[9] eq "nodes" ) {
 $nodes = $moveData[10];
 }else {
 $nodes = $moveData[9];
}
print "$_[0]\n";
print "$moveData[2]\n";
$dbh->do("INSERT INTO moves(match_id,move_nr, /
black_move,fen,best_move, /
player_cp,nodes,depth,penalty) VALUES($_[1],$_[2], /
$_[3],'$_[4]',' /
$_[5]','$moveData[7]','$nodes','$moveData[2]','$_[6]')");
}

#SUB ROUTINE THAT CREATES A NEW BATCH
sub batch_creation {

my $batchInsert = $dbh->prepare("INSERT INTO batch(/
batch_desc,batch_name,max_moves,max_time_w, /
pen_type_w, pen_dist_w,pen_range_w,distribution_w /
,pen_mean_w, / max_time_b,pen_type_b, /
pen_dist_b,pen_range_b, /
distribution_b,pen_mean_b) VALUES /
(?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
$batchInsert->execute($_[0],$_[1],$_[3],$_[4], /
$_[5],$_[6],$_[7],$_[2], /
$_[8],$_[4],$_[9],$_[10],$_[11],$_[2],$_[12]);
print "Batch $_[1] successfully created\n";
}

#SUB ROUTINE THAT LIST BATCHES
sub batch_listing {
if ( $_[0] ) {
my $batchSelect = $dbh->prepare("SELECT * FROM /
batch WHERE batch_id = ?");
$batchSelect->execute($_[0]);
```

```
while ( my @row = $batchSelect->fetchrow_array() ) {
print join(" ", @row);
}
print "\n";
}        else {

my $batchSelect = $dbh->prepare("SELECT * FROM batch");
$batchSelect->execute();
while ( my @row = $batchSelect->fetchrow_array() ) {
print join(" ", "@row\n");
 }
 }
}

#SUB FOR PARSING INPUT STRING FROM "-P" switch

# W_fixed:W_dist:W_range:W_mean:B_fixed:B_dist:B_range:B_mean
# example: fixed penalty (300 ms) for black, none for white
# -P "0:::fixed:::300"

# example: Uniorm noise, mean 300 ms, range /
100 ( in practice somewhere between 200ms /
and 400ms ) for white, fixed 200ms for black
# -P "variable:uniform:100:300:fixed:::200"

sub parsePenaltyString {

my @penarray = split /:/,$PENALTY_STRING;

$WHITE_PENALTY_TYPE = $penarray[0];
$WHITE_PENALTY_DIST = $penarray[1];
$WHITE_PENALTY_RANGE = $penarray[2];
$WHITE_PENALTY_MEAN = $penarray[3];
$BLACK_PENALTY_TYPE = $penarray[4];
$BLACK_PENALTY_DIST = $penarray[5];
$BLACK_PENALTY_RANGE = $penarray[6];
$BLACK_PENALTY_MEAN = $penarray[7];
}

#SUB FOR CALCULATING PENALTY
sub calculatePenalty {
my $penSelect = $dbh->prepare("SELECT pen_type_w, /
pen_mean_w,pen_type_b,pen_mean_b /
FROM batch WHERE batch_id = ?");
$penSelect->execute($_[1]);


my @row = $penSelect->fetchrow_array();
if ( $player eq $WHITEIP ){
print "Penalty given to WHITE: $row[1]\n";
if ( $row[0] eq "fixed" ){
#  my $random_number = int(rand($randRange));
#  print "Rand numb: $random_number\n";
my $penTime = ($_[0] - $row[1]); #+ $random_number;
print "Current movetime with penalty WHITE: $penTime\n";
return $penTime;
}
}
if ( $player eq $BLACKIP ){
print "Penalty given to BLACK: $row[3]\n";
if ( $row[2] eq "fixed" ){

#my $minimum = $randArray[1];
my $random_number = int(rand($randRange));
print "Rand numb: $random_number\n";
my $penTime = ($_[0] - $row[3]) + $random_number;
print "Current movetime with penalty BLACK: $penTime\n";
return $penTime;
```

```
 }
 }
}

sub usage {

print "Usage:\n";
print "-h for help\n";
print "-v for verbose ( more output )\n";
print "-d for debug ( even more output )\n";
print "-B create (CREATE needs the following switches: -e /
<batch description> -n
<name> -D <distribution> -x <Matchcount> -P <(for W&B) /
type:dist:range:mean>-m <MaxMoves> -t
<MaxTime>(Must be defined in MS ex: 1000)\n";

print "-B list (LIST needs the following switches: /
-i <ID of batch to list>\n";
print "-p play (need to supply the following switches: -B /
<name of batch> -d <search depth> OR -M <Movetime> /
-w <White player IP> -b <Black player IP>)\n";
}

sub verbose {
print "VERBOSE: " . $_[0] if ( $VERBOSE or $DEBUG );

}

sub debug {
    print "DEBUG: " . \$_[0] if ( $DEBUG );

}
```

Analysis framework script

Analysis framework - analysis.pl

```
#!/usr/bin/perl
use strict vars;
use Getopt::Std;
use IO::Socket;
use DBI;

#DATABASE VARIABLES#

 #MySQL Server HOST IP ADDRESS
my $DBHOST = "*";
#MySQL Server PORT
my $DBPORT = "13306";
 #The database to use /
my $DB     = "*";
when connecting to the server
#The user in which is used to login with
my $DBUSER = "*";
#Password of the current user
my $DBPASS = "*";

#SWITCHES NEEDED FOR THE SCRIPT#

my $opt_string = "vhA:c:b:d:i:r:t:";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

my $VERBOSE           = 1 if $opt{'v'};
my $DEBUG             = #1 if $opt{'d'};
#Switch for supplying which BATCH should be analyzed
my $BATCHID           = $opt{'b'};
#Switch for batch creation or listing
```

```
my $ANALYSIS_BATCH    = $opt{'c'};
 #Switch Analysis batch ID in order to LIST a specfic batch
my $ANALYSIS_BATCH_ID = $opt{'i'};
my $REFTYPE           = $opt{'t'};
#Specify a depth to run the chess engine
my $DEPTH             = $opt{'d'};
#Switch for inputing the referee IP
my $REFIP             = $opt{'r'};
#To initiate the analysis process
my $ANALYZE           = $opt{'A'};
my $move_id;
my $match_id;
my $fen;
my $best_move;
my $move_depth;
my $move_cp;
my $move_nodes;
my $move_time;
my $player_cp;
my $move_bestmove;
my %moveData;
my $socket;
 if ( $opt{'h'} ){
 usage();
 exit 0;
}

#CONNECTING TO MYSQL DATABASE#

my $dbh = DBI->connect("DBI:mysql:$DB;host=$DBHOST; /
port=$DBPORT",$DBUSER,$DBPASS) or die $DBI::errstr;
if ( $dbh ){
print "Connection to $DBHOST and $DB successfull\n";
}


#Checking if correct swithces are provided with the script
usage() and die "Need to supply correct switches \n" unless /
$ANALYSIS_BATCH or $ANALYZE;

#BATCH CREATION AND BATCH LISTING#
#Code for batch creation and batch listing
 if ( $ANALYSIS_BATCH && $ANALYSIS_BATCH =~ /create/ /
 && $BATCHID && $DEPTH && $REFTYPE ) {
 print "Creating new analysis batch\n";
 batch_creation($BATCHID,$DEPTH,$REFTYPE);
} elsif ( $ANALYSIS_BATCH && $ANALYSIS_BATCH =~ /
 /list/ && $ANALYSIS_BATCH_ID ) {
 print "Now listing wanted batch(es)\n";
 batch_listing($ANALYSIS_BATCH_ID);
} elsif ( $ANALYSIS_BATCH  =~ /list/) {
 print "Now listing wanted batch(es)\n";
 batch_listing();
}

if ( $ANALYZE && $ANALYZE =~ /analyze/ && /
$ANALYSIS_BATCH_ID && $REFIP && $DEPTH) {
print "Analysis commencing with the following referee
$REFIP\n";

my $new_analysis = 1;
my $analysis_going = 1; #IF true(1) analysis will /
keep running, if set to false(0), the analysis will stop

while ( $analysis_going ) {
#  if ( $new_analysis == 1) {
#  $dbh->do("INSERT INTO analysis_result /
(anal_batch_id,ref_id) VALUES($ANALYSIS_BATCH_ID,' /
$REFIP')");
# }
fetch_data();
run_analysis($REFIP);
}
# Sub routine that initiates the chess match between /
two virtual machines running an instane of stockfish.
sub run_analysis {
my $analyzer = $_[0];
# Connecting to a socket on virtual machines hosting /
#a chess server.
$socket = new IO::Socket::INET (
PeerAddr  => $analyzer,
PeerPort  =>  3333,
Proto => 'tcp',
) or die "Couldn't connect to Server\n";

#my $lastline = "";
if ( $socket ){
# 1. Send brett
my $lastMove = 0;
print "Current MOVEID and FEN is being evaluated $move_id /
 AND $fen\n";
print $socket "position fen $fen\n";
# 2. beregn trekk
print $socket "d\n";

print $socket "go depth $DEPTH\n";
# my $bestmove; #The best possible move
while ( my $line = <$socket> ) {
 chomp $line;
 #print "$line\n";
 if ( $line =~ /info\sdepth\s(\d+)\sscore\smate\s(\d+)/ ) {

$dbh->do("INSERT INTO analysis_result(anal_batch_id, /
match_id,move_id,ref_id,/original_move,prop_move,/
prop_cp,prop_nodes,move_depth,movetime,optimal_cp, /
optimal_nodes,cp_loss,optimal_move,optimal_time) /
VALUES($ANALYSIS_BATCH_ID,$match_id,$move_id,' /
$REFIP','$best_move',0,0,0,0,0,0,0,0,0,0)");
$lastMove = 1;
}
elsif ( $line =~ /info\sdepth\s(\d+)\sseldepth\s(\d+)\ /
sscore\s(\bcp|mate)\s(-{0,1}\d+)\snodes\s(\d+) /
\snps\s(\d+)\stime\s(\d+)\smultipv\s(\d+) /
\spv\s(\({0,1}\w{0,4}\){0,1})/ ){
#print "$move_depth\n";
#print "$move_cp\n";
add_result($1,$4,$5,$7,$9,$fen);
$lastMove = 0;
}
if ( $line =~ /bestmove/ ){
last;
}
}
 if ( $lastMove == 0 ) {
 re_run();
}

#print $socket "position fen $fen moves $bestmove\n";
$new_analysis=0;
print "Conversation finished\n";
print "socket still active\n" if $socket;
close($socket);
 }
}
}

```

```
sub re_run {
 foreach my $move ( keys %moveData ){
 my $lastline = '';
 my $analysisDepth = $DEPTH - 1;
 print $socket "position fen $moveData{$move}{'Fen'} moves /
 $moveData{$move}{'Bestmove'}\n";
 print $socket "d\n";
 print $socket "go depth $analysisDepth\n";
 my $finished = 0;
  while ( not $finished and my $line = <$socket>  ) {
  chomp $line;
  # print "$line\n";
  $lastline = $line if ( $line =~ /info depth/ );
   if ( $line =~/bestmove/ ) {
   print "THIS IS LASTLINE: $lastline\n";
   add_optimalData($lastline,$move);
   print "Setting finished flag\n";
$finished = 1;
#                        last;
 }
 }
}
}
 # }

#my $optimalSwitch;
sub add_optimalData {
my $nodes;
my $move = $_[1];
my @optimalData = split(' ', $_[0]);
  if ( $optimalData[9] eq "nodes" ) {
$nodes = $optimalData[10];
}else {
$nodes = $optimalData[9];
    }

print "$_[0]\n";
print "$_[1]\n";
print "$nodes\n";
print "$optimalData[7]\n";
print "$optimalData[17]\n";
print "$optimalData[13]\n";
#if ( $best_move eq $moveData{$move}{Bestmove} ) {
my  $optimalSwitch = $optimalData[7] * -1;
#}
my $cpDiff = $optimalSwitch - $moveData{$move}{CP};
#my $absDiff = abs($cpDiff);

$dbh->do("INSERT INTO analysis_result(anal_batch_id, /
match_id, move_id,ref_id,original_move,prop_move, /
prop_cp,prop_nodes,move_depth,movetime,optimal_cp, /
optimal_nodes,cp_loss, optimal_move,optimal_time) VALUES /
($ANALYSIS_BATCH_ID,
$match_id,$move_id,'$REFIP','$best_move', /
'$moveData{$move}{Bestmove}',
$moveData{$move}{CP},$moveData{$move}{Nodes} /
,$moveData{$move}{Depth},$moveData{$move}{Time},/
$optimalData[7],$nodes,$cpDiff,'$optimalData[17]', /
$optimalData[13])");
}

sub add_result {
print "adding move for $_[4]\n";
$moveData{$_[4]}{"Depth"}=$_[0];
$moveData{$_[4]}{"CP"}=$_[1];
$moveData{$_[4]}{"Nodes"}=$_[2];
$moveData{$_[4]}{"Time"}=$_[3];
$moveData{$_[4]}{"Bestmove"}=$_[4];
```

```
$moveData{$_[4]}{"Fen"}=$_[5];
}

#FETCH MOVE DATA FROM MOVE TABLE.
#hvis verdien er mindre enn 0 ganger med -1. Trekk CPene /
#fra hverandre, hvis det er mindre enn 0, gang med -1
#absolute value perl
sub fetch_data {

# empty moveData
%moveData = ();

# TODO Lag read lock og hent latest_id i tillegg til batch_id
# Getting batch id from analysis_batch in order /
#to fetch the correct set of matches
$dbh->do("START TRANSACTION");

my $batchSelect = $dbh->prepare("SELECT batch_id,latest_id /
FROManalysis_batch WHERE anal_batch_id = ? FOR UPDATE");
$batchSelect->execute($ANALYSIS_BATCH_ID);
my @row = $batchSelect->fetchrow_array();
my $batch_id = $row[0];
my $latest_id = $row[1];
print "THIS IS LATESTID $latest_id\n";
print "This is the batchID: $batch_id\n";

my $analysisData = $dbh->prepare("SELECT moves.move_id, /
moves.match_id,moves.fen,moves.best_move,player_cp /
FROM moves LEFT JOIN matches ON  /
moves.match_id = matches.match_id WHERE moves.move_id > /
'$latest_id' AND matches.batch_id = ? ORDER BY /
moves.move_id ASC LIMIT 1");
my $countResult = 0;
$analysisData->execute($batch_id);
while ( my @row = $analysisData->fetchrow_array() ) {
$countResult++;
print "COUNTRESULT: $countResult\n";

if ( $row[3] =~ /(none)/ ) {
my $analysisData = $dbh->prepare("SELECT moves.move_id, /
moves.match_id,moves.fen,moves.best_move,player_cp /
FROM moves LEFT JOIN matches ON /
moves.match_id = matches.match_id /
WHERE moves.move_id > '$latest_id' /
AND matches.batch_id = ? /
ORDER BY moves.move_id ASC LIMIT 1");
}

$move_id = $row[0];
$match_id = $row[1];
$fen = $row[2];
$best_move = $row[3];
$player_cp = $row[4];

$dbh->do("UPDATE analysis_batch SET latest_id = $move_id, /
@last_update = NOW() WHERE /
anal_batch_id = $ANALYSIS_BATCH_ID");
}
$dbh->do("COMMIT");

unless ($countResult) {
print "No more moves to fetch - Going to sleep for /
10 seconds\n";
exit 0;
#  fetch_data();
}

#TODO Hent move_id where move_id > latest_id ... /
```

```
Update anylis_batch med latest. Deretter release lock
#Limit 1 Order by (Desc eller asc)
}

#SUB ROUTINE THAT CREATES A NEW ANALYSIS BATCH
sub batch_creation {

my $batchInsert = $dbh->prepare("INSERT INTO /
analysis_batch(batch_id,depth,ref_type) VALUES (?,?,?)");
$batchInsert->execute($_[0],$_[1],$_[2]);
print "Analysis batch $_[0] successfully created\n";
}

#SUB ROUTINE THAT LIST ANALYSIS BATCHES
sub batch_listing {
 if ( $_[0] ) {
 my $batchSelect = $dbh->prepare("SELECT * /
 FROM analysis_batch
 WHERE anal_batch_id = ?");
 $batchSelect->execute($_[0]);
  while ( my @row = $batchSelect->fetchrow_array() ) {
  print join(" ", @row);
  }
 print "\n";
}else {

my $batchSelect = $dbh->prepare("SELECT * FROM /
analysis_batch");
$batchSelect->execute();
while ( my @row = $batchSelect->fetchrow_array() ) {
print join(" ", "@row\n");
}
}
}

sub usage {

print "Usage:\n";
print "-h for help\n";
print "-v for verbose ( more output )\n";
print "-d for debug ( even more output )\n";
print "-c create (CREATE needs the following switches: /
-b <Id of batch to analyze> -t <Type of referee> /
-d <Depth at which to analyze> <)\n";
print "-c list (LIST needs the following swtiches: /
-i <ID for listing specific analysis batches \n";
print "-A analyze (need to supply the following switches: -r /
 <IP of referee> -i <ID for which analysis batch to use>)\n";
}

sub verbose {
    print "VERBOSE: " . $_[0] if ( $VERBOSE or $DEBUG );

}

sub debug {
    print "DEBUG: " . $_[0] if ( $DEBUG );

}
```