NTNU – Trondheim
Norwegian University of
Science and Technology

Distributed Hosting of Systems using donated Computer Resources

André Skoglund Hansen

Master of Science in Computer Science
Submission date: June 2013
Supervisor: Svein Erik Bratsberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

To host a value-added internet service, like a web page with a large user base, an organization either has to rely on cash donations or it has to monetize the service. The monetization of the service often means degrading the quality of the service or making it less appealing. This is why this project introduces a new *business model* where services can be run by the users themselves by letting them donate computer resources. This again should lower the operating cost of the service. The new business model is introduced by developing a framework that allows developers to implement their services in a way that let dedicated users participate in hosting the service. First the framework was developed, and then the framework was used to develop an example implementation of a distributed web page. For it to be realistic that users would be able to partake in an operation like this, a project goal was to make sure that the technical demand from users are low. The framework is written with this in mind and the reached simplicity is presented at the end of the report.

Sammendrag

For å drifte en internet tjeneste, for eksempel ei nettside eller et flerspiller spill med en stor brukerbase, er en organisasjon i de fleste tilfeller avhengig av donasjoner eller nødt til å tjene penger på tjenesten. Ofte så involverer det å tjene penger på en tjeneste at man forringer kvaliteten på tjenesten eller så gjøres den mindre tilltrekkende ved en betalingsmodel. Derfor handler dette prosjektet om å introdusere en ny modell for å kunne drifte en tjeneste, en modell hvor brukerne av tjenesten drifter tjenesten selv. Dette gjøres ved å la brukerene donere maskinressurser som kan brukes til å drifte tjenesten. Derfor er det i dette prosjektet utviklet et rammeverk for hjelpe utviklere i å implementere tjenesten på en distribuert måte. I tilegg så er det et poeng at for at brukere skal kunne bidra, så er det en fordel at prosessen med å bidra er teknisk enkel. Dette er vurdert oppnådd og er presentert til slutt i repporten.

Preface

A student master thesis at the Norwegian University of Science and Technology (NTNU) for André Skoglund Hansen, and is the conclusion to the Game Technology specialization at the Department of Computer and Information Science (IDI) under the Faculty of Information Technology, Mathematics and Electrical Engineering (IME). The general project idea spawned from the project member and was narrowed down after guidance from the supervisor.

I would like to thank my supervisor Svein Erik Bratsberg for accepting my master thesis project and for theoretical guidance during the project work phase. I also would like to thank two of my fellow students at Gribb: Damian Dubicki and Ole Jørgen Rishoff for all those long nights working side by side, for the hours of watching and analyzing Dota 2 streams, and for feedback and contribution to my master thesis. And lastly, I would like to thank myself for hanging in there.

Content

List of Figures

Part I,

Introduction

1 Project Background

In this chapter the motivation, goal and context behind the project is given.

1.1 Motivation

Today the business model for most web based services either involves payment or degrading the quality of the product to monetize it. The degrading can involve limiting the functionality, adding advertisement or monetization of the user base and its data itself. These are all unwanted solutions. Another option is to rely on cash donations from organizations or persons. A last option is that service owners fund the service themselves, however, not viable for large and costly systems. This is a reality I want to challenge: I want to examine if it is possible to reduce services operational cost by using crowdsourcing as a business model to operate them. Or in other words, I want to examine if it is possible to develop a technology that allows users to donate their computer's resources to operate a foreign service.

Personally, this idea and motivation spawned from the experience of playing numerous games that had to degrade the quality of the game to survive. Examples of this were commercial games that used pay-to-win models and non-commercially games that could not evolve its functionality because the required computer resources for said functionality were not affordable.

1.2 Project Goal

The end goal is to develop software that can assist developers at developing their own services that can easily be distributed hosted by end users. "Easily" is an important keyword, as the higher the technical requirements to partake in the distribution, how fewer users will partake. A sub-goal of this is to consider what the potential using crowdsourcing to host services is. There already exist many scientific projects that use users' computer resources to do calculations for them, which are projects that is worth looking into.

A second sub-goal is to investigate what technologies can be used, and a third sub-goal is to derive inspiration from similar software that exists today.

1.3 Project Context

A student master thesis at the Norwegian University of Science and Technology (NTNU), conducted as a conclusion to the Game Technology specialization at the Department of Computer and Information Science (IDI) under the Faculty of Information Technology, Mathematics and Electrical Engineering (IME). The general project idea spawned from the project member and was narrowed down after guidance from the supervisor.

1.4 Stakeholders

This chapter identifies all stake holders of this project.

1.4.1 Project member

- André Skoglund Hansen

The project member is concerned with understanding relevant theoretical principles and interested in exploring technical solutions that can be part of a finished product. Finally the project member has a strong interest in delivering a solid project report as it defines the master thesis grade.

1.4.2 Supervisor

- Svein Erik Bratsberg

The supervisor’s main concern is the quality of the documentation, and additionally has interest in the result part as it can be a start point for new master thesis projects that take the problem domain further.

1.4.3 Users: Developers

While the user group is not an active part of the project, developers of systems that require large amounts of computer resources will have an interest in the conclusions of this project.

1.4.4 Users willingly to donate their computer resources

The last group of stake holders is users that are willingly to donate their computer resources to do work for a service. These users are interested in working with a simple system and getting some form of acknowledgement for doing so.

2 Research

As part of the project process, a set of research questions have been posed. These questions will help further defining the project goal. The methodologies used to answer them are then described afterwards.

2.1 Research Questions

Allowing new worker nodes to join an already running system in a plug and play fashion, to then partake in hosting it requires complicated features and techniques. These research questions tries to break down this complication into isolated problems which can be answered and possible be done prototyping on.

RQ1: Is this business model applicable to the real world?

RQ1.1: What are the alternative business models?
RQ1.2: Is there similar business models which are good enough?

RQ2: How can distributed hosted web systems best be done?
RQ2.1: How can distribution of static files best be done?
RQ2.2: How can distributed request processing best be done?
RQ2.3: How can distributing a database best be done?

RQ3: How should a distributed network be operated?
RQ3.1: How should new worker nodes connect/disconnect to the network?
RQ3.2: What architecture(s) should the distributed network have?

RQ4: How can it be avoided that evil workers do damage to the overall system?
RQ4.1: Can it be avoided that an evil worker does permanent damage to the persistent storage?
RQ4.2: Can sensitive data be stored at untrusted worker nodes?
RQ4.3: Can processing be done in a fashion that makes it hard for an evil worker to do damage?

2.2 Research Methodology

A general description of the methods used in this project is given in the following subchapters.

2.2.1 Literature Review

Literature review consists of reading articles, studies, books or other form of documentation about technology or subjects relevant to the project. The literature review is a most important during the projects theoretical pre-study phase, but also relevant during the design and implementation phase.

2.2.2 The Engineering method

The engineering method is a process where you observe existing solutions, propose better solutions, build/develop a solution, measure and analyse the solution, and repeat the process until no more improvements appear possible within a given timespan. It is one of the methods Basili [1] describes as part of the scientific method.

3 Development

This chapter describes which development methods have been considered for the project, which have been used, and which tools have been used.

3.1 Development Methods

A development method is a process that can be used to structure, plan and control the development of the system. During this project the two development methods Waterfall Model and Scrum was considered, but due to the theoretical nature of the project it was deemed unnecessary. The technical research question has instead been answered by literature review and by prototyping.

3.1.1 Software Prototyping

Software prototyping consists of developing working, but incomplete software, which focuses on testing concepts or critical functionality. Examples are; testing technology for performance, testing architectural decisions and testing user interfaces. In addition, prototyping can help discovering problems earlier, problems that could be severe enough that the architecture or technologies have to be changed or scrapped.

3.2 Development Tools

The following tools have been used to either develop a prototype, test the prototype, or to write the various documents during the master thesis.

Eclipse is an Integrated Development Environment (IDE) that can be used to write and manage software code.

PuTTY is an open source terminal emulator. In this project it has been used to control an external server that has been used to test the master node.

Dropbox is a file hosting service used to store personal files in the cloud, it can be used as an external backup service or to share files between computers.

GitHub is a web-based hosting service that can be used to store project source code files. It is built upon Git [2] which is a distributed version control system.

Google Docs is Google’s office suite and has been used to create word and excel type of documents.

Microsoft Word 2010 is a word processor program.

Part II,

Pre-study

4 Concepts

As a basis for the paper a set of concepts and techniques has to be discussed.

4.1 Business Models for value-added services

A value-added service is a telecommunications industry term that describes services that is built on top of and adds value to an existing service by increasing the usage of it [3], this case the existing service is the internet.

To offer any sort of value added service, like a web page, an application server or a web service there will always be a cost associated with it. To cover this cost the service provider can use one or more of the following business models.

4.1.1 Free

The cost will have to be covered by the administrators of the service themselves. Often this means that persons closely involved with the system cover the cost. In other cases organizations offer the service for free because they can, like Google with Gmail and Google Drive. However, organizations may indirectly benefit from offering popular services like this. The benefit can for an example be a positive name branding effect or the possibility of doing data mining on user data generated by the service.

4.1.2 Free + Premium

Services can be offered for free to gain traction/popularity, while at the same time dedicated users can pay for a premium account with extra or improved functionality. These users are able to cover the total operation cost. An example of this is Dropbox [4] where users either can have a free account with a small amount of storage, a pro account with a large amount or storage, or business plan aimed at businesses with several user accounts.

4.1.3 Crowdsourcing

Often a service cost is covered purely by the generosity of its users, and this generosity can take one of the following forms.

Users **donates money** to support the service, Wikipedia, one of the most popular web pages in the world [5] is funded this way.

User **donates computing power** to the service. This is within this project’s scope and no cases where this are done have been discovered. The only similar cases are volunteered computing with two subcategories: 1) Donating computer resources to scientific projects. 2) Donating computer resources to a middleman which sells the resources to a third party and give a percentage of the revenue to a selected charity. These two cases are discussed later.

4.1.4 Advertisement

Advertisement is a business model mainly suited for services that provide content; sound, image, video and text to users. The cost is covered by charging a third party for advertisement views.

4.1.5 Micro-transactions

In computer games a micro-transaction is a small real-life payment a user can do to gain an in-game token such as items, visual effects or access to new content. It is an increasingly popular business model proved to work and is the only source of revenue for League of Legends, one of the most played online game today [6][7]. It is not uncommon that this payment model is combined with other payment models.

Often this business model can take shape as a **pay-to-win** system, and if this model is used in a multiplayer game, you end up with a system where players can pay for an advantage over other players. If these advantages get to large, the game will feel unfair for the players not willingly to spend real life money.

4.1.6 Subscription Fee

A subscription fee is a regular payment and can range from weekly to yearly. This can be a problem if it has to compete against services that are free, as a one-time payment makes the service less appealing to first time users.

4.1.7 One-Time Payment

The user pays a one-time payment to gain access to a service. The one-time payment model faces the same problem as the subscription fee model.

4.2 Web Hosting

There are several methods to host an internet service, but in generalized form there exists two main categories:

- Dedicated hosting, which involves renting a dedicated server, either a virtual or an actual server that the renting part has either limited or full (root) access to.
- The second is cloud or clustered hosting, which allows for much easier scaling as the available computer resources can be scaled up.

Hosting web pages at home is rarely done for web pages in production, and it is even common that ISPs has policies that forbid this. For example: the translated terms of usage for Telenor's private subscriptions states that "*It is not allowed to set up servers connected to the Telenor broadband connection for commercial activities*" [8]. It is also common that ISPs block port 80 for incoming traffic to private subscribers. This is probably done because ISPs do not want people to use the full extent of their internet connection.

4.3 Decentralized System

A decentralized system is a system without a central organ in control, and still works in unison for a common purpose. In computer technology a fully decentralized system is described by Nelson Minar and Marc Hedlund as a system where “not only is every host an equal participant, but there are no hosts with special facilitating or administrating roles” [9].

4.4 Relevant Application Architectures

Not all application architectures are relevant to this project; the architecture has to allow the application to stay decentralized. In this chapter we take a look at those who are relevant.

4.4.1 Peer-to-Peer (P2P)

“*Peer-to-peer is a class of applications that takes advantage of resources – storage, [CPU] cycles, content, human presence – available at the edges of the internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer must operate outside the DNS and have significant or total autonomy from central servers*” [10].

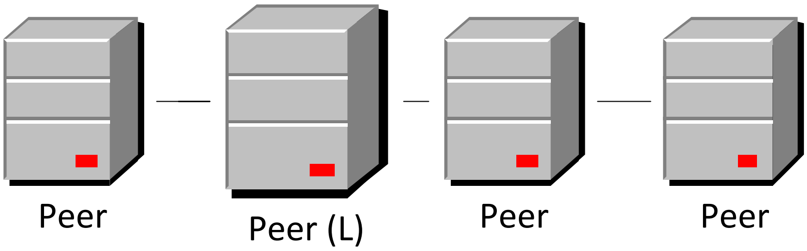

Figure 4-1: A flat peer-to-peer hierarchy, where one peer is selected as leader (L).

Peer-to-peer computing uses computers volunteered by generous users to do distributed computing, and are mainly found in scientific computing projects. A non-scientific example of this is the BitTorrent technology, which takes advantage of users storage and network throughput, bur do not depend on low latency.

4.4.2 Master-Worker (MW)

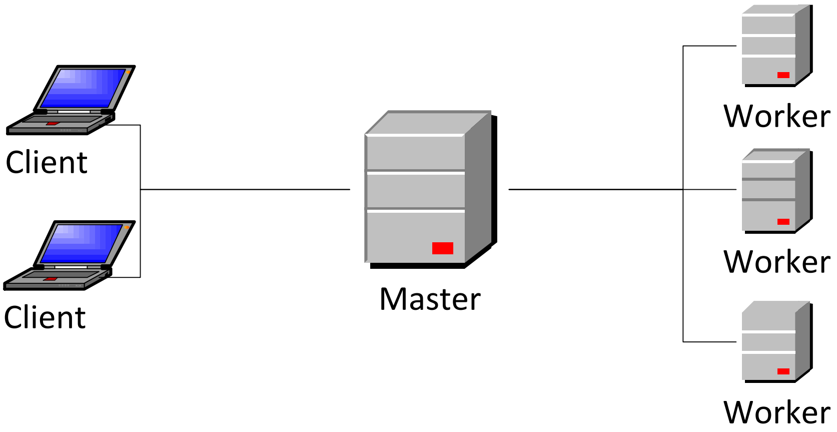

Figure 4-2: Master-Worker Architecture

Instead of having a flat hierarchy like in a peer-to-peer network, master-worker architecture allows one participant to be the authority. This has the benefit of being a simpler protocol where work distribution, process management and fault tolerance all belong to one master. A downside is that the master is in danger of becoming the performance bottleneck as the amount of workers increase [11].

If the tasks done by a master consists of more than just routing traffic, for an example tasks that involves heavy processing. It could also be beneficial to implement another layer of hierarchy, where a super node administrates all master nodes, and master nodes administrate a limited group of worker nodes. This super-master-worker hierarchy can be seen on Figure 4-3.

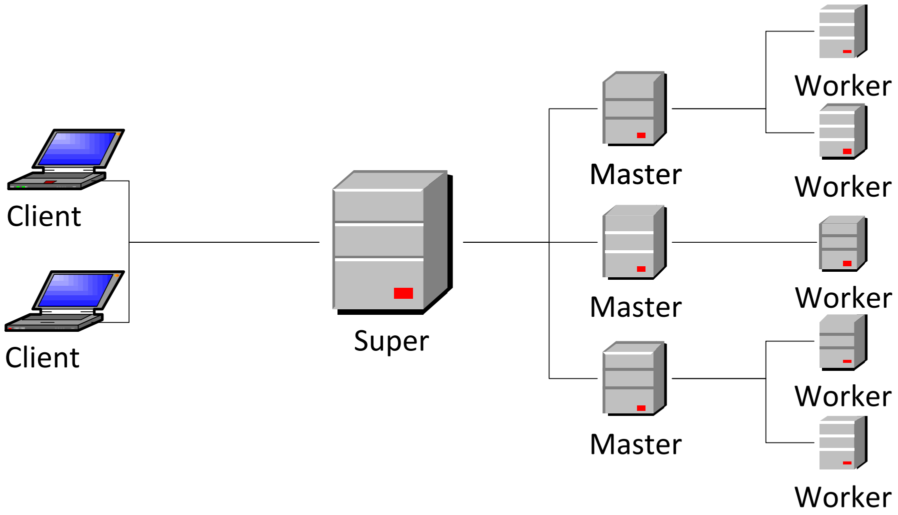

Figure 4-3: Super-Master-Worker architecture

4.4.3 *Service Oriented Architecture (SOA)*

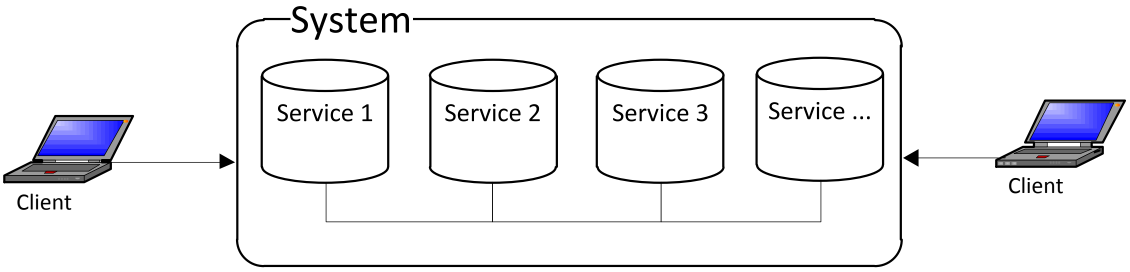

Figure 4-4: A simple representation of a SOA

In a SOA there exist decoupled services which alone serve a specific role, that collectively provide the complete functionality of a large and complex system. Services can access other services using their API and data is most commonly exchanged using either XML or JSON, which are language-independent and human readable data formats. This makes the services easier to tailor together and reuse. Since no restrictions are posed by the service and data formats themselves, applications using the service only need to be able to parse strings.

SOA does not replace P2P or MW architecture; instead it may be considered as a secondary architecture that the P2P or MW architecture can be built on top of.

4.5 Distributed Systems

A distributed system is a system where multiple computers work together over a network. The network can either be locally close (LAN) or globally separated (WAN). Often specific software is written to delegate tasks in a distributed manner. In the next chapters a few distributed techniques that have been considered are presented.

4.5.1 *Distributed Database*

The most central part of a distributed system is the distributed storage of data, which can be done by either using:

- Replication: all data is replicated among all nodes
- Fragmentation: data is split up and scattered to different nodes. This again is done either by splitting data horizontally or vertically. In horizontal fragmentation the split is done between database tuples, for an example; if you create two Facebook profiles one of them is stored at location 1 and the other at location 2. In vertical fragmentation the tuples are split on schema level, for an example; all your Facebook “likes” relations are stored at location 1 and the rest of your profile is stored at location 2.

4.5.2 *Process Migration*

To achieve parallel computing over a distributed system the *process migration* technique can be used, which is the act of transferring a process between two machines (the *source* and the *destination* node) during its execution [12]. This should not be confused with *remote invocation* which is the creation of a process on a remote node. The most relevant benefits - which are mentioned by [12] - of process migration is:

- **Dynamic load distribution**, by migrating processes from overloaded nodes to less loaded ones.
- **Improved system administration**, by migrating processes from the nodes that are about to be shut down or otherwise made unavailable.
- **Data access locality**, by migration processes closer to the source of the data.

Process migration can be more complex than simply transferring the state of the process to the destination node; it can also involve sharing other resources like memory. Naturally, the effectiveness of shared memory is limited to the quality of the internet connectivity between the two nodes, latency and bandwidth.

A shortened version of the migration steps, based on [13] are:

1. The process state is extracted from the source node.
2. A destination process instance is created at destination node.
3. State is transferred and imported into the new instance.
4. The new instance is resumed.

4.5.3 Content Deliver Network (CDN)

A CDN is a distributed system of servers which main goal is to serve content to end-users on the internet with high availability and performance. It achieves this by replicating the content to servers across the world. The content served is usually static content like text, html, sound, video, scripts, documents, etc. However, some CDN’s can deliver dynamic content. CDN is primarily an acronym that describes a business model where companies offer value-added service providers a way to host their static content.

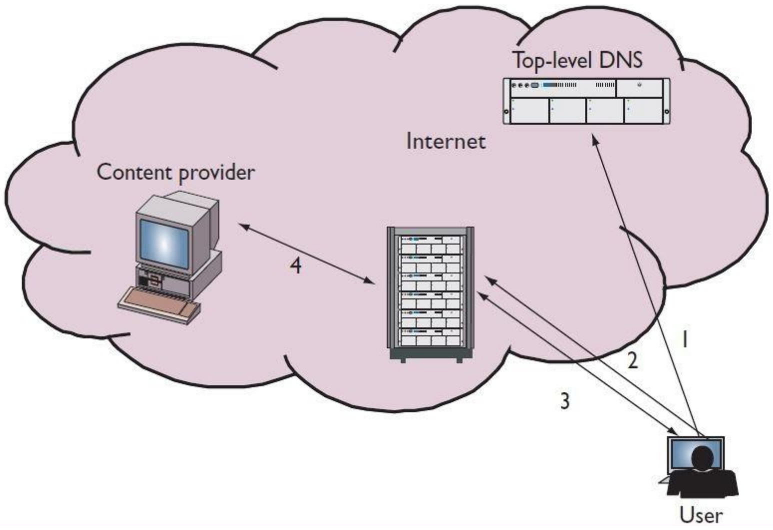

Figure 4-5: A client http content request [14] using a CDN

Secondarily CDN describes a technique that is illustrated in Figure 4-5. The CDN works as a buffer between the content provider and the users, and will grab content on demand (step 4) and return it to the user (step 3) as well as cache it. Caching it allows the CDN to respond directly to the consecutive requests for the same content, which lowers the stress on the content provider's server.

4.5.4 Load Balancing

Load balancing is one of, if not the most central part of creating scalable services. Load balancing is the method of distributing workload across available resources. A resource can be a disk drive, a CPU, a whole computer or anything that can be replicated. In this project using load balancing to create scalable web pages has been the research focus.

H. Bryhni, E. Klovning, and Ø. Kure [15] describe that most web links are accessed using its canonical name (CNAME, example: www.example.no) instead of the 4-byte IP address. Then using the Dynamic Naming System (DNS) the CNAME is resolved to an IP. With replicated HTTP servers, load sharing requires the ability to map one logical address onto several different physical servers. This mapping can be done at three logical places; at the client, among the servers, or by the network. Five methods are here summarized. Method 3 is a software method, while method 4 and 5 is a hardware method.

#1 Remapping at the Client - Transparent

Consider a case where a system is hosted by several replicated servers with their own unique IP, and they are all mapped to the same CNAME. Not that DNS provides a distributed database for mapping between CNAME and IP addresses**.** So when a client access a CNAME URL, the client will query its local domain's name server for an IP mapped to that CNAME. This local name server will then lookup that CNAME progressively through the DNS until it finds the *end* name server which has the hosted servers IPs mapped to the CNAME. This end name server will return one of the IPs in the manner of round robin. Now the local name server has **one** IP mapped to the CNAME, which it caches and now every consecutive local CNAME lookup points to that IP. The result of this will be an unevenly distributed system if many clients have the same local name server, as they all get the same IP. Why? Because the local name server only stored one IP as seen on Figure 4-6 and a round robin distribution is not possible.

This method is considered to be remapping at the client since the CNAME and IP mapping is finally saved at the client. However, one could also argue that it is remapping at the DNS.

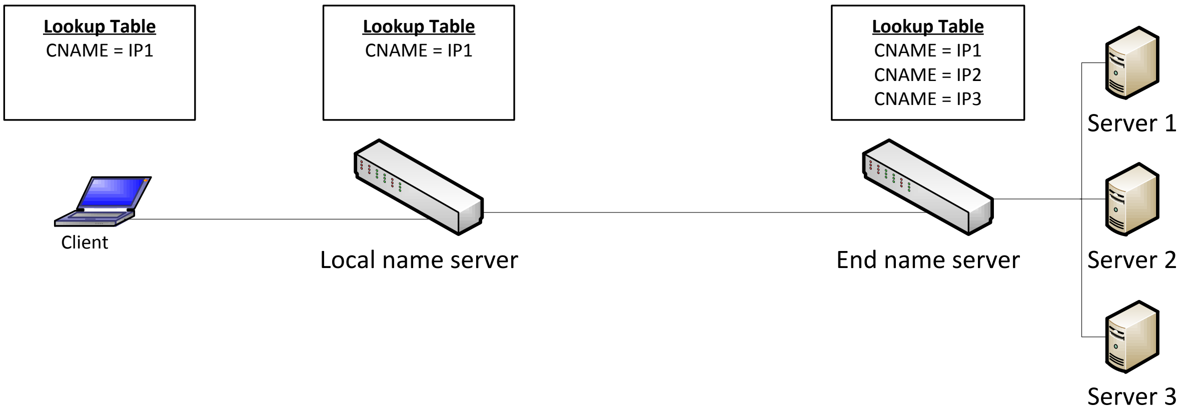

Figure 4-6: How the CNAME-IP mapping propagates through the network

#2 Remapping at the Client - Non-Transparent

An example of a non-transparent remapping is how Netscape load balanced requests to their homepage. Netscape had its browser periodically query a DNS with random number X between 1 and 32 using homeX.netscape.com to select one available server to load pages from. This effectively spread the load between the 32 servers [ref]. This solution is rather unique and was only possible because Netscape had such a huge percentage of the browser marked.

#3 Remapping in the Server

This is a software method that uses a master web server to process the HTTP requests and distributes them on to one of the replicated servers. The drawback is that remapping is done at application level, and the request must traverse the full protocol stack four times before the request is actually processed. Two times down, and two times up the stack. In most cases the master server will be the bottleneck, but it is still a possibility if the requests are time demanding, and that the HTTP process overhead is only a small part of the full process time.

#4 Remapping in the Network - at the network layer

In this method a system has several replicated servers, where each has a unique IP address, and the delegation is done by a *HTTP Scheduler*. The HTTP scheduler is in practice - but not necessarily physically - between the replicated servers and the web client as it intercepts all IP packets to the system's logical address (for example www.example.com). The interception is done by having the HTTP scheduler's IP mapped to www.example.com at DNS level. It distribute requests by inspecting the intercepted IP packets (at the network layer) and changing the destination addresses to the address of the replicated server with the lowest load and then sending the packets out again. Since the HTTP scheduler will receive the packets for the processed HTTP request from the chosen replicated server, it also has to change the destination address of the processed IP packets to the web client's address and send the packets out again.

This method do not require, but works better if the HTTP scheduler is close to the replicated servers, so network capacity is high and load information polled from the replicated servers are accurate. Since every bit of data has to pass through the HTTP scheduler it will have to handle the sum of the system's network traffic.

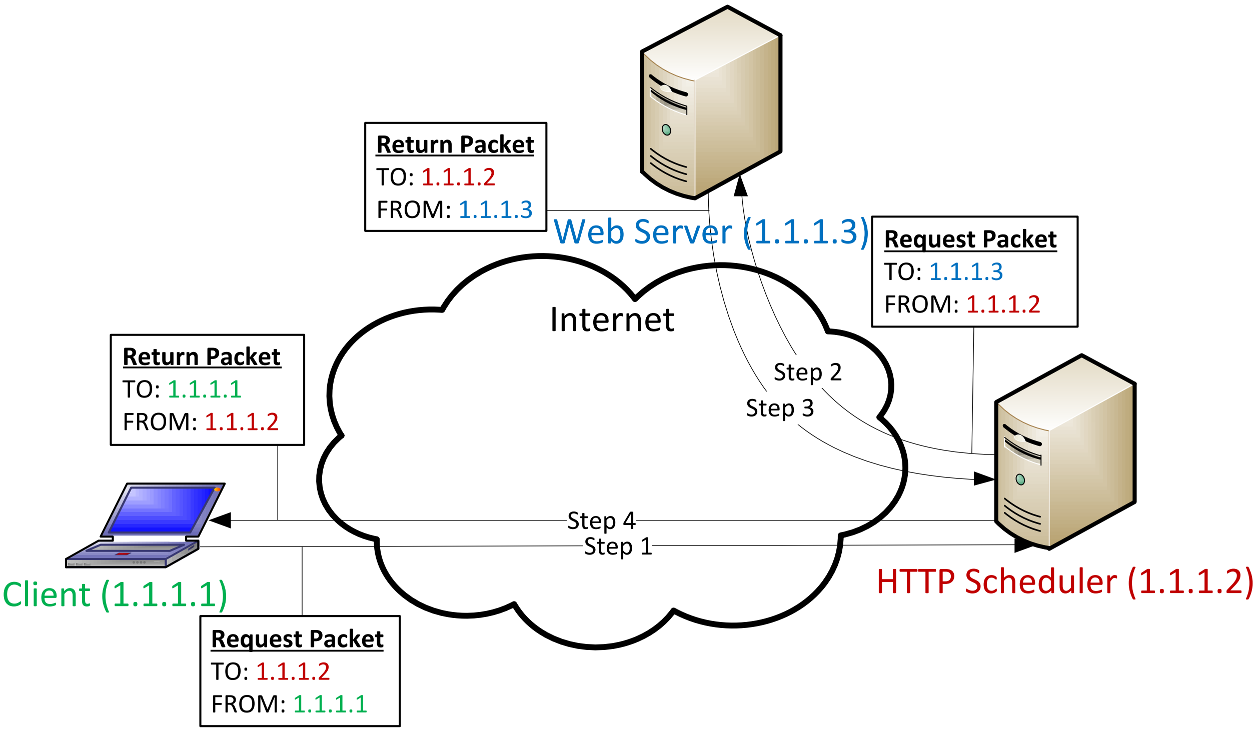

Figure 4-7: Load balancing using remapping at the network layer

Figure 4-7 show how a HTTP request packet is processed through the internet and the system. As it can be seen, the scheduler has to inspect every packet the system receives, and needs to maintain a table that contains information that make sure the packet at step 3 is changed to the correct client IP (step 4). Conceptually, one could say that the scheduler works as a network component and not a web server. The reason the web server cannot send the return packet to the client directly, is that the client would not recognize the web server as the correct responder, which is the scheduler, and would drop the packet unopened.

#5 Remapping in the Network - between the network and link layer

In this last method all replicated servers have the same IP as the logical server, and now the HTTP Scheduler has to inspect the IP packets and remap by another label, like the MAC-address (link layer) or port. However, since both these cases would result in the scheduler not doing any changes on the network layer, it would end up sending all packets to all replicated servers as they all have the same IP. To counter this, the scheduler has to hardcode its Address Resolution Protocol (ARP) table, a table that maps network layer addresses to link layer addresses.

The advantage of this method is that IP packets do not have to be modified. The disadvantage is that all replicated servers have to be on the same subnet, a disadvantage that is a deal breaker in conjunction with this project.

Scheduling Algorithms

In the cases where there is a scheduler who is responsible for distributing load, the scheduler has to use a scheduling algorithm. A few worth mentioning are:

Round-round: distributes requests evenly one at a time for every replicated server. A problem with this algorithm is that requests may have different processing times. So a unlucky scenario is that a replicated server might not be finished processing the last request and then get a new one, while at the same time a second replicated server has zero requests waiting.

Least connections: distributes requests to the replicated server with the least active connections.

5 Technologies

Several technologies and software that might be relevant for this project or could be used for inspiration have been researched.

5.1 Nginx

Nginx pronounced “Engine X” is a HTTP and reverse proxy server, as well as a mail proxy server [16]. Currently Nginx is recommended to be used on Unix bases operative systems, and discouraged to be used with Windows.

5.1.1 Nginx Load Balancer

Nginx has a module named *ngx_http_upstream_module* [17] that when activated allows the system administrator to specify multiple servers that Nginx will distribute requests to using a technique called reverse proxy. This proxied method is the same as the load balancing method #3 [18], which is explained in chapter [4.5.4]. A few key properties of the Nginx load balancer:

- Can use the round-robin or least connections scheduling algorithm.
- Can distribute requests between servers based on client IP addresses, so sessions are maintained. This is done by mapping client IPs to worker nodes IPs, and making sure all requests from the same client are forwarded to the same worker.

- Can add new servers to the server pool without restarting [19]. To add a new worker node to the sever pool a configuration file has to be updated and then reloaded. The reload does not require the Nginx process to restart and current requests can go unaffected.
- Can weight servers; some servers can get more traffic directed to than other.
- The load balancer only works on unix based operative systems.
- Detects failing services by a configurable *fail_timeout* parameter, given in seconds. Servers will in combination with the configurable *max_fails* parameter be removed from the load balancing loop when the server has failed enough times.

5.2 Haproxy

Haproxy [20] is an alternative to the Nginx load balancer module. While Nginx is a web server that can also serve as a load balancer, Haproxy's only functionality is load balancing. It does not perform noticeable better [21] but is together Nginx the two software load balancers that are most commonly used.

The most interesting difference is that Haproxy allows a maximum amount of connections to be configured for each worker node. This option help avoid that a very long running request in a round robin queue props up the request queue for the unlucky worker node that received it.

5.3 Apache ZooKeeper

ZooKeeper [22] is a distributed open source coordination service for distributed applications. It helps developers at developing distributed systems by simplifying administrative tasks such as synchronization of data and coordination between nodes out of the box. ZooKeeper can also be used to develop client applications, and will then create the architecture seen on Figure 5-1.

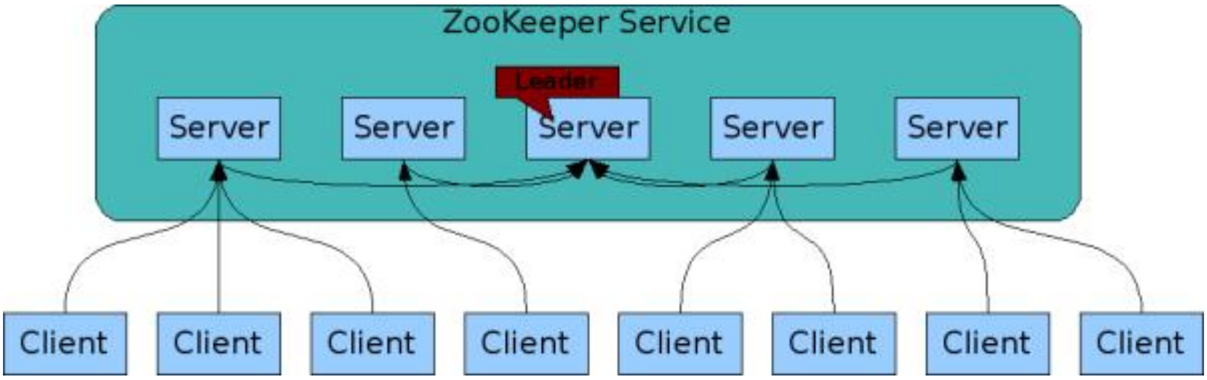

Figure 5-1: ZooKeeper architecture [22]

Some qualities of ZooKeeper:

- There is no single point of failure, as the ZooKeeper application is replicated over a set of hosts. A leader is selected using a leader election algorithm, and quickly replaced if it goes down.
- It is designed with the idea that clients connects to a single ZooKeeper server and maintains a TCP connection which it uses to send requests, get responses, listen to events and send heart beats. In other words, this is not a load balancer for HTTP request.

5.4 Protocol Buffer

Google developed Protocol Buffer because they needed a "*language-neutral, extensible way of serializing structured data for use in communication protocols, data storage and more*" [23]. By defining the data structure in a .proto file you can let java, python or c++ classes be automatically generated by the Protocol Buffer compiler. An example of a data structure is given:

```
message Person {
      required string name = 1;
      required int32 id = 2;
      optional string email = 3;
}
```

When compiling this with the protocol buffer, a corresponding Java, Python or C++ Person class and PersonBuilder class are created. These classes can then be used to create objects which are serialized for further use, like communication or storage. Note that Protocol Buffer does not take part in communicating or storing the serialized data. Individual code has to be written for this.

5.5 MongoDB and NoSQL

From their own homepage: "*MongoDB is an open-source document database, and the leading NoSQL database*" [24]. It was developed with the purpose of being *agile and scalable*.

A NoSQL database uses simpler storage and retrieval mechanisms than SQL databases, which makes them better suited for scalable systems. This is because NoSQL uses a looser consistency model, where storage is simply a key-value record. This also increases performance as inserts and updates do not need to be validated according to a database scheme.

MongoDB has the following relevant properties:

- Focus on easy data replication.
- **Auto-sharding:** Sharding distributes a single logical database system across a cluster of machines.

5.6 Twisted

Twisted is an event-driven network engine written in Python and is licensed as open source under the MIT license [25]. It is a framework that assists developers in creating custom network applications by letting developers write callback functions for network events. A Twisted network application has two distinct components, a server and a client. And the most central network events for the components respectively are:

- connectionMade – A client connected / Connected to Server
- connectionLost – Lost connection with client / Lost connection to Server
- dataReceived – String data received from client / server.

5.7 Node.js

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [26]. It is a very young (created in 2009) web server software that has become popular because it synergies very well with new technologies like WebSocket [27] and MongoDB and because it allows the backend to be written in the same language as the frontend.

5.8 Universal Plug and Play (UPnP)

UPnP is a set of network protocols that allows one network device to discover, access and possible change other network devices. The intention is to simplify the process of installing network devices that might require specific network options activated. The interesting part of this protocol is that it allows software installed on a user’s computer to access the router the computer is behind and change the NAT configuration so specific ports are routed to the user computer.

6 Related Systems

Projects similar to this project have been studied for two reasons; for inspiration and to understand if there is, and if so, how much potential may lie in crowd-driven systems.

6.1 Case Study: BOINC

Berkeley Open Infrastructure for Network Computing (BOINC) [28] is the most known software for volunteered computing. It allows people to donate their home computers free CPU time to one of the many available scientific projects. Most projects are named under the @home standard, and among them is the

SETI@home project which in 2008 had harvested more than a million CPU years that was roughly estimated to be worth more than $1 billion [29].

For all the BOINC projects combined there are currently (May, 2013) 2.5 million registered users and 380 thousand active computers participating. These are impressive numbers and on top of this; these projects are scientific projects which users have no real gain from supporting, except the common benefit from scientific progress. The only incentive users have to participate are gaining *virtual credits* which are added as a score to their profile, team and country. One can only speculate how willingly users are to support a service that they use themselves. More interesting statistics about BOINC can be found at http://boincstats.com/.

As described on their homepage: *"BOINC is designed to support applications that have large computation requirements, storage requirements, or both. The main requirement of the application is that it be divisible into a large number (thousands or millions) of jobs that can be done independently."* [30]. Each job to be done are queued up and feed to the BOINC worker nodes in a First Inn First Out (FIFO) order. Worker nodes receive jobs and execute them when they are idle. A worker node's idle status is defined by time since last user input for that computer. This makes it hard to use BOINC for any real-time processing. As there is no way to control that only *running* worker nodes receive jobs and to secure that a worker node running a job is not stopped by a user input.

A problem with volunteered computing is how the results generated by the users can be trusted. Most BOINC projects solve this with cross-checking, which involves sending multiple users the same tasks and only accepting data that are validated to be equal among multiple job calculations. Another way that this can be solved in BOINC is to write custom validators that validate user generated results by specific conditions.

6.2 Case Study: Superdonate

Like BOINC, Superdonate [31] allow generous users to download a client that uses the CPU when it otherwise would idle. The CPU will then be used to do calculations that require little data, but could take long time to finish. The difference from BOINC is that there is a third party that pays Superdonate for CPU time, and that the client chose a charity that a percentage of the third party payment goes to.

As part of this master thesis Superdonate was contacted multiple times about their usage numbers. Numbers that could further indicate the potential of donated computer resources. No reply was received, so no real conclusion could be taken. Nevertheless, the concept is still interesting.

6.3 Case Study: Amazon Elastic Compute Cloud (Amazon EC2)

There exists 10s if not 100s of cloud computing providers, one of them is Amazon EC2 [32]. Cloud computing providers allow system administrators to quickly scale the resource capacity of the services they are administrating. Amazon EC2 allows this scaling to be done through their EC2 web service which can be accessed either through a web page or an API. Three payment models can be used:

- **On-Demand Instances**: Pay for compute capacity by the hour, where compute capacity can be increased or decreased depending on the application demand. This model is best suited for applications with short term, spiky or unpredictable resource requirements.
- **Reserved Instances**: Upfront payment for a compute instance. Best suited for applications with long term and predictable resource requirements. Such as popular web pages.
- **Spot Instances**: The user sets a maximum hourly price he is willing to pay, if there are available instances to this price or lower the application will run. The current spot price fluctuates based on supply and demand. This model is best suited for applications with flexible start and end times.

It is hard to give any exact numbers on the cost of using a cloud computing payment model, as the pricing is defined by the applications resource usage. It will vary from application to application. However the reserved instances model is naturally the most expensive one and the spot instances model the cheapest.

Amazon EC2 actually provides 750 free hours of instance run time each month, and anything beyond this will be charged. In a hypothetical scenario where a web server process requests in an average time of 200ms per request, the web server could serve 5.2 requests per second. This is a total of 13.5 million requests per month for free.

6.4 Case Study: Google App Engine (GAE)

Google App Engine do like Amazon EC2, they provide a cloud computing service. However it is a bit more restrictive. Whereas Amazon EC2 provides a virtual OS, GAE provides a programming language environment where the developer has to develop his application according to a specific API for that language. The current main languages that can be used are Java and Python and the experimental ones are PHP and Go.

Like Amazon, Google has a payment model that scales by the resource usage of the application. Each Google developer account can have 10 applications that each can have 1 GB storage and a CPU and bandwidth to support an efficient app serving around 5 million page views a month [33]. Anything beyond this is charged for. Registering a developer account requires a unique phone number, this serves as a defense against persons attempting to game the system by registering multiple accounts.

6.5 Free Hosting Services

Free hosting services do exists, but they either have a low storage capacity, low bandwidth capacity, enforce ads or have limited or no OS access where they only can be accessed through web admin pages.

Part III,

Own Contribution

7 Proof of Concept “Plug N Host”

To understand the possibilities and limitations of a system that distributes load in a ‘*plug n host*’ fashion a proof of concept have been developed.

The concept can best be described from a developer’s view-point. Imagine that a developer wants to develop a system which is a value-added internet service that requires an extraordinary amount of resources. Resources can be CPU, memory, storage, etc. The project has no real potential revenue due to its nature and it is impossible to afford paid hosting. However, the project might have many dedicated users that might want to support the system by donating their computer’s resources. The developer can build the system by extending “Plug N Host”, a framework that comes with tools to distribute a system in a way that allows workers to join in on processing in a plug n play fashion, thereby the Plug N Host name. The coupling between the framework and the custom implementation using the framework can be seen on Figure 7-1. The gray box illustrates the framework and the green boxes the custom implementation. These colors are used in all related figures.

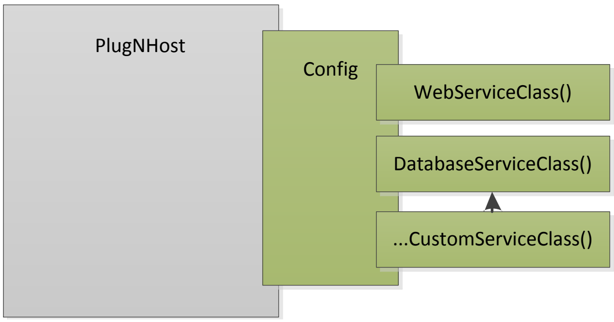

Figure 7-1: A simple representation of a custom framework usage.

To test this concept both the core functionality for PlugNHost and a custom example project with a config file and an HttpServiceClass have been developed. Further details on the configuration will be described later.

In a best case scenario the user that wants to donate his computer only has to 1) Download the custom software. 2) Run a script that installs dependencies and starts the service. One thing that has to be pointed out, the worker node has to be publicly accessible, meaning it has to have the service specific ports open and accessible from the internet. This poses a big limitation on which users can participate, since it either requires that the users’ computer is behind a router that supports UPnP or it increases the technical skill required from users.

7.1 Stages and functional requirements

This concept was developed in five stages where functionality has been implemented incrementally.

7.1.1 Stage 1: Web Page

F1 – Online Web Page	A web page is hosted by a production web server, and is then accessible using a browser.
F2 – Spawnable Web Server	The web server process should be spawnable programmatically, using a python script.

7.1.2 Stage 2: Load Balancing

F3 – Manual single-worker distribution	Serve the web page with a single worker using the load balancing software.
F4 – Manual multi-worker distribution	Serve the web page with 2 or more workers using the load balancing software.

7.1.3 Stage 3: Automate

F5 – Master-Worker Communication	The master and workers can communicate with each other over a stable communication protocol.
F6 – Automated multi-worker distribution	If a worker joins or leaves the network the load balancer has to automatically adjust itself and serve requests to the available workers nodes without manual changes.

7.1.4 Stage 4: Configurable

F7 – Software is configurable	To allow for reuse of software and easier implementation of custom solutions, custom classes have to be defined in a configuration file.

The F7 functionality is in other words first step in creating a framework.

7.1.5 Stage 5: Monitor

F8 – Monitor distribution at master	Requests, requests distribution and other data which can help investigate performance should be logged and presented live.

7.2 Technology Rationale

The rationale behind choosing the technology for the core and for the custom implementation is given in this chapter.

7.2.1 Core Technology

For the core functionality there have been made two technology decisions. The first one was which language to do the implementation in; the only requirement for the language was that it has to be cross-platform. Since the framework has to be cross-platform to make runnable for as many users as possible. Python, a cross-platform programming language was a natural choice as the project member had enough experience with it to make sure it did not pose as an extra obstacle between reaching the project goals.

The second decision was how communication between the master and the worker was going to be done. Either a socket based communication module could be written from scratch or already existing software could be used. Among existing software ZooKeeper and Twisted was considered. ZooKeeper was discarded as it tries to do much more than was needed, for an example its purpose is not only to assist in communication between the server parts (master and workers), but its purpose is also to be used as a protocol between clients and servers, and to provide synchronization mechanism which is better suited for peer-to-peer architectures. It was undesirable to further complicate the framework with functionality not necessary to create a working proof of concept. ZooKeeper also couples service logic tighter with distribution logic, more than was desired. In fact, ZooKeeper is better considered as an alternative to PlugNHost, an alternative that could be used if a client-server application was to be developed. Instead Twisted was chosen as its only purpose is to assist in communication at a high level. It is an event-driven networking engine, which means that the engine triggers a defined callback method for a few predefined network events. Currently, the only communication done is a notification from the worker to the master when a service is available and ready to accept load. However it is perceivable that future versions could include communicating administrative data, like server load, list of current workers or similar status data. These are all communication events that could demand action taken on the receiving end. That is why Twisted is a good choice with its event-driven design. It is also a programming design pattern that is simple to understand and implement.

It should also be mentioned that BOINC was considered. However BOINC's architecture goes against real-time processing of requests, like a web requests would require. Even if the limitations of the BOINC could be surpassed by clever techniques it was decided early on to be dropped as potential software as this project would be an unnatural area of use.

7.2.2 Custom Technology

When it comes to the development of the example project that uses the framework, a custom implementation of a distributed web page has been written. This implementation uses Nginx as a load balancer at the master node and Nodejs to process the web requests at the worker nodes. Node.js was chosen because it allow web servers to be started using a shell command. This makes it easy to start new worker node instances for testing purposes.

Nginx was chosen for the example project and is recommended to be used in all custom implementations requiring a load balancer, because it performs well and has a good reputation. The benefit of using a software load balancer like Nginx over a load balancing method that takes use of the DNS is that the worker node list can be updated much quicker. If the DNS was used to load balance, a joining/leaving worker node IP would propagate slowly through the DNS which in turn could be a problem. If a client has mapped a disconnected worker node to the CNAME it would get a dead response, and it is undesirable that joining workers aren't able to accept load until the DNS is updated. Especially if there is a tendency that users don't donate their computer resources for longer amount of times.

8 Architecture

In this chapter the architecture of the proof of concept is described and illustrated with a process and logical view. The rationale behind the architectural decisions is also given.

8.1 Physical Architecture

The network topology of the proof of concept is a true master-worker architecture, an architecture that are described in chapter [4.4.2].The reason a master-worker architecture was chosen over a p2p architecture is that there is no easily imaginable way a p2p architecture can guarantee the administration rights and ownership of the system to one worker node. In a p2p system the peers must have a *significant or total autonomy from central servers*. If there is no central server that all requests are channeled through, there would be nothing that stops an evil worker from processing requests outside the systems scope, potentially stealing or destroying them.

Even if a hybrid master-p2p architecture was chosen, where one worker cleverly keeps administrative rights through a special protocol implemented at code level there would still not be anything that stops the evil worker from ignoring this protocol if the software was reverse-engineered and re-implemented with an evil intention.

8.2 Logical View

The logical view should describe how the functional requirements have been met at code-level, using class diagrams.

8.2.1 Class Diagrams: Core Classes

A core class is a class which are used by and common among all custom implementations. It should not be thought of as a library, as a library is something you import and use in projects. In practice there are two different types of core classes:

- A Master and Worker service manager class, named MasterNode and WorkerNode respectively. These two manager classes read the configuration file which contains information about service classes and runs custom class behavior:
  - Master: Runs the on_worker_change() event.
  - Worker: Runs the start() and stop() event.
- Abstract classes that should be extended by the custom classes:
  - BaseWorkerService – a class that should be extended by each worker service class.

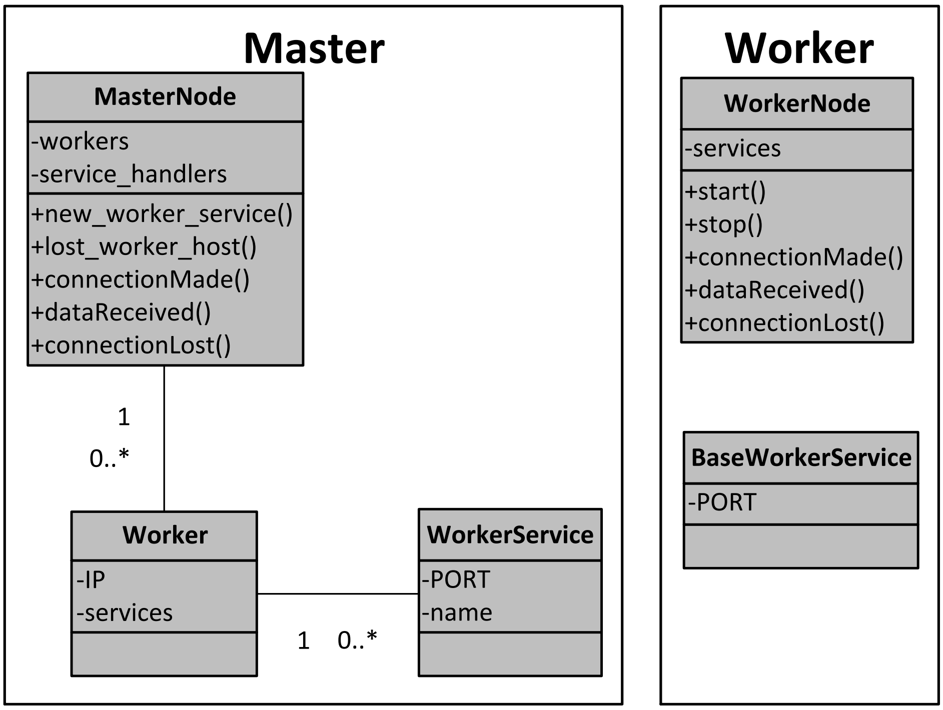

Figure 8-1: Core classes, stripped for some variables and methods names.

The MasterNode and WorkerNode class extends Twisted's protocol.Protocol class. This gives them the connectionMade, dataReceived and connectionLost methods which are communication events.

8.2.2 *Class Diagrams: Example Custom Classes*

Each different project that uses PlugNPlay will have to implement its own custom service classes. An analogy of this is that the core classes are the pole of a flag pole, while the flag fabric itself represents the custom classes.

Every service that should be distributed has to have a *service class pair* defined in the configuration file. This service class pair consists of one class that is run on the master node and one that is run on the worker node. As seen Figure 8-2 worker service classes have to extend a base class called BaseWorkerService which is a class that implements some functionality required to communicate with the master. Among this functionality is the notify_ready() method that should be run by the extending class when it is ready to accept load. In the custom implementation the extending class is the NodwejsWorkerService.

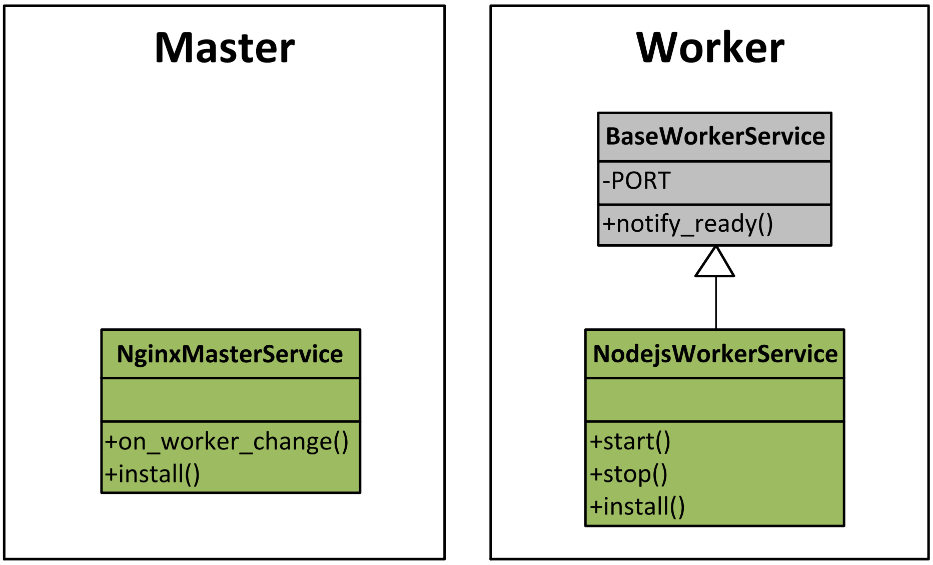

Figure 8-2: NginxMasterService and NodejsWorkerService, a service class pair

As part of this project install and startup scripts have been written in python, these help simplify the process of installing and starting the master and worker node. This simplification requires all service class pairs to implement an install() method which should contain custom logic to install dependencies for each service.

As a result, every custom service requires five events callbacks to be written, two (1, 2) in the master class and three (3, 4, 5) in the worker class. These events are summarized:

1. Master, on_worker_change(): The amount of workers changed.
2. Master, install(): Will be run by the install script at the master node.
3. Worker, start(): Will start the service.
4. Worker, stop(): Will stop the service.
5. Worker, install(): Will be run by the install script at the worker node.

It is imaginable that some custom implementations would require other events than start and stop that needs to be communicated between the master and workers. That’s why future versions of the framework should allow the implementation of custom events that can be triggered and listened to on both the master and worker.

8.2.3 Configuration

The configuration file is a python file and requires two main configuration definitions:

- **COMMUNICATION_PORT**: A port used for administrative communication.
- **SERVICES**: A list of service pair classes. In the example below (Figure 8-3) one service have been defined, the “http” service. This service has two required configuration parameters, the MasterClass and the WorkerClass. These configurations should be dotted class path strings that are relative to the configuration file. So normally it would be something like “folder1.folder2.file.class”, while it is now only “class”. Take a look at the GitHub project for a complete example of this. The GitHub project url is given under result.

```
# TCP port to be used for twisted communication
COMMUNICATION_PORT = 7999

# Created services, each item represent one service and the master and worker
# class contains business logic for the specific services.
SERVICES = {
    'http': {
        'MasterClass': "NginxMasterService",
        'WorkerClass': "NodejsWorkerService",
    },
}
```

Figure 8-3: An example configuration.

8.3 Process View

A process view should present how the system behaves at runtime and how the different parts of it communicate.

8.3.1 Service Oriented (Process) Architecture

The PlugNPlay framework encourages a SOA by letting the developer to split his functionality into different service classes. These service classes are registered in the configuration file, which the framework reads and runs in each its process.

The reason a SOA is enforced is that it is a good architecture to use with distributed systems. Consider the case with a dynamic web page where users upload a lot of static content like pictures. In this case it makes sense to divide up the system in three services, one static content service for hosting pictures, one web server service and one database service. Now the distribution can be split in two, where a worker either hosts a static content service or runs the dynamic http request processer which consists of a web server service and a database service. This split allows the worker nodes to be specialized, instead of just being a full replication of the system.

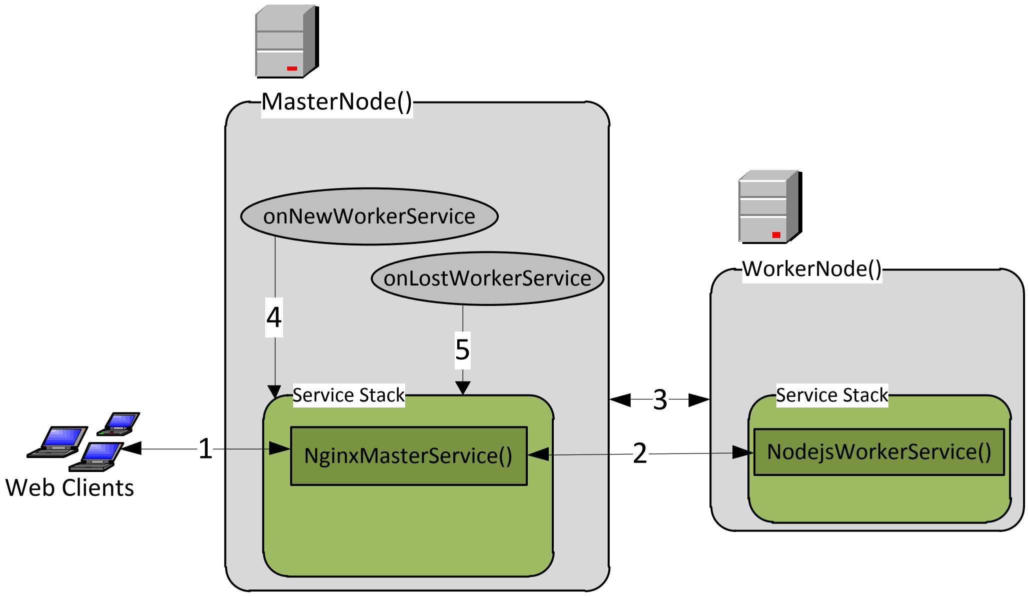

Figure 8-4: The service oriented architecture of the framework

In this project a working distributed web page has been developed. Figure 8-4 depicts this, gray parts are core functionality and green parts are custom functionality. The numbers indication communication parts:

1. Users send http requests.
2. The NginxMasterService() uses Nginx as a Load Balancer that forwards http requests to one of its registered worker nodes.
3. Communication between the master and the worker is done using Twisted.
4. When a new worker joins the system it will notify the master about which services it is hosting, this triggers the new_worker_service() method that forwards the notification to the specific service class, in this example the NginxMasterService() class. The NginxMasterService() class then behaves according to the custom implementation.
5. When a worker node either terminates his participication or the connection is lost, the lost_worker_host() method is triggered, this method will also forward a notification to all the master service classes affected. Since every worker node can run multiple services, the framework mighty have to notify multiple master service classes.

8.3.2 *Starting Services*

Services are started by the framework according to the classes specified in a configuration file that is read by both the MasterNode and WorkerNode. In Figure 8-5 it can be seen how they together start one service pair, NginxMasterService and NodejsWorkerService respectively. In the custom example these service classes main purpose is to start a child process, which is the best way to do this as the services then can be implemented completely decoupled from the framework. The PlugNHost framework should only contain protocols for administrating the services.

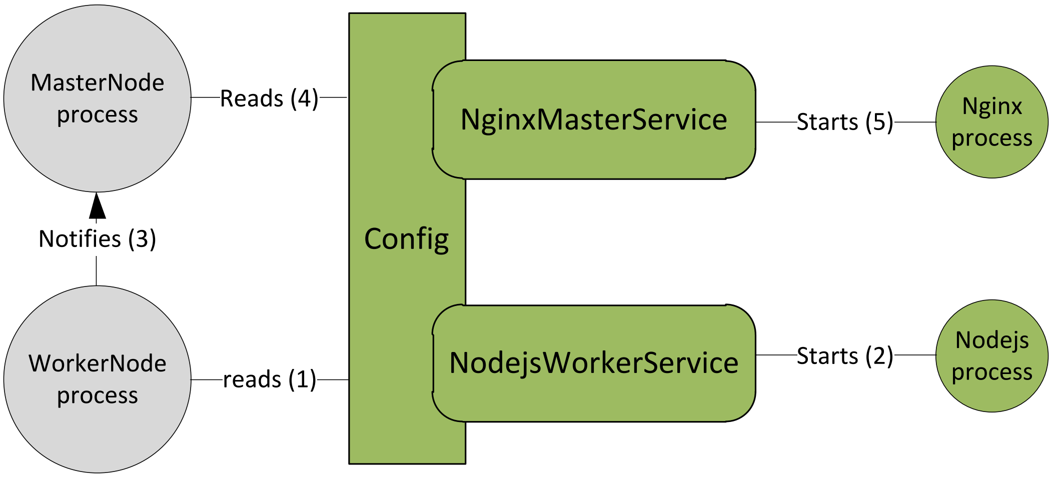

Figure 8-5: The five steps of starting a service at the worker and master node.

A property of the framework is that services can be started on demand and without restarting the MasterNode or WorkerNode process. This is easily doable because Python is an interpreted language and code is executed directly, in contrary to precompiling code where this would be a complicated procedure. So in a scenario where the developer wants to add a new service without interfering with the already online services, the developer can update the configuration file and start the service at a worker node (step 1+2). The worker would then notify (step 3) the master that it is ready to accept load. If the master service that is paired with the newly started worker service is running, it will update it with the new worker node list, otherwise it will attempt to start it (step 4+5). By reloading the configuration file before step 1 and 4 one allows the configuration file to be updated live, and since the configuration file contains a path to the service classes they can also be loaded live. It should be noted that in the current state of the framework, services cannot be deleted or updated, only added.

When Twisted detects that a worker node have disconnected it will trigger the connectionLost event, which will update the available service workers list.

8.3.3 Integrating with existing systems

The framework should be fully decoupled from the implementation of the service itself. Only the callbacks for events that are listed in chapter [8.2.2] has to be implemented to tie the business logic for distributing the service (the framework) with the service. On Figure 8-5 the “Nginx Process” and “Nodejs Process” represent the implementation of the service.

9 Evaluation and Conclusion

As a summarization of the paper the research questions are attempted answered.

RQ1: Is this business model applicable to the real world?

To answer this question one has to look at related systems that exist today, a few of them are listed at [6]. If a developer wants to host a dynamic web page both GAE and Amazon EC2 are viable options. However, once the web page reaches a few millions hits per month the issue of cost will arrive. To put this in perspective, in a worst case scenario the free cap for GAE and Amazon EC2 might be reached at 2 million hits. This would require 500 users to do 5 hits each hour to reach the cap, or 4 users to do 11 hits each minute. If the estimated maximum amount of hits per month is 12.5 million as the Amazon EC2 calculations suggest you still get a cap that is reached with 5 hits each second.

Another point is that the mentioned scenario does not take into consideration server load spikes. And it is realistic that spikes will occur since it is perceivable that users of a service are more prone to use it at certain times of the day and Amazon EC2 does not deliver free load balancing to multiple instances.

For small to maybe medium services, using Amazon EC2 or similar free cloud hosting is probably viable, but once a service should be able to handle a medium or larger user base this cannot be considered viable anymore. GAE allows for more free processing power in total due to its 10 free projects, but this does still not serve as a universal solution as GAE only allow specific language implementations to be run.

During the research phase it was discovered that some ISPs have policies that prevent private internet subscribers from hosting web pages or other internet services. This poses an interesting problem, how will this policy affect a private person that donates their home computer as a worker node? Some ISPs don’t allow commercially operated servers on their internet connections, other don’t allow any server activity. Most likely this will be a non-issue for several reasons:

- ISPs do not generally take action against private persons that have servers that do not do very excessive internet activity.

- PlugNHost only uses obscure ports, and not known ports like the HTTP port 80, which makes it less likely to be detected.
- Most systems using a framework like PlugNHost are not going to be commercial.

Also, the goal of this business model is not only to allow people to donate their home computers as worker nodes, it is also to allow people to donate all of or part of the processing power of their running dedicated servers.

RQ1.1: What are the alternative business models?

The alternative business models have been described in chapter [4.1].

RQ1.2: Is there similar business models which are good enough?

For some resource demanding services, relying on free hosting, which can be unreliable, or relying on donations from users is simply not enough. So the answer would be no. Providing an alternative business model option where users instead donate computer resources can lower the threshold for donating. As mentioned SETI@Home had harvested donated computer power estimated to be worth more than 1 billion dollars and this to a cause of finding extraterrestrial life. It is impossible to imagine that this sum could have been achieved with traditional cash donations from the same user base.

RQ2: How can distributed hosted web systems best be done?

Chapter [4.5] answers this, and the most relevant technique is load balancing.

RQ2.1: How can distribution of static files best be done?

For commercial services, paying for a CDN would be the best way to distribute the serving of static files. CDN is described in chapter [4.5.3]. For non-commercial services using its own load balancing network is a viable option.

RQ2.2: How can distributed request processing best be done?

There are two ways to do this, either using load balancing or process migration. Process migration is an option that better fit for heavy calculations that requires long running processes. For web requests which are required to be processed quickly, load balancing is the way to go. In chapter [4.5.4] several load balancing methods are discussed.

RQ2.3: How can distributing a database best be done?

There are two types of distribution of databases that can be done, replication and fragmentation. These methods are discussed in chapter [4.5.1].

RQ3.1: How should new worker nodes connect/disconnect to the network?

In the PlugNHost concept framework a possible solution to this is presented, see chapter [8.3.1] and [8.3.2].

RQ3.2: What architecture(s) should the distributed network have?

A distributed network can have a peer-to-peer architecture or master-worker architecture, and in addition to either of the two it can be built using a SOA scheme. Which architecture should be chosen depends on security and ownership concerns, and is discussed in chapter [8.1]

RQ4.1: Can it be avoided that an evil worker does permanent damage to the persistent storage?

This is a question that has not been answered in this paper. The question is therefore listed under future work. However it is safe to say **no** if the database is distributed using a fragmentation method without any replication. Since that means one worker sits on the only copy of any given part of the data.

RQ4.3: Can processing be done in a fashion that makes it hard for an evil worker to do damage?

The case study of BOINC revealed that this was solved by cross-checking data between multiple clients. So in the case of processing web requests one can send the web requests to two worker nodes and validate that the returned web responses are equal. Keep in mind a damaged worker response might not only be the result of an evil intention, it can also be the result of hardware failure that somehow damaged the response. Since it is possible that two clients return a damaged response, the level of security can be increased by increasing the amount of workers used to cross-check between.

10 Demonstration of Result

The project goal is to “*assist developers at developing their own services that can be easily distributed hosted by end users*”. This can be demonstrated by downloading and testing the custom framework implementation. As it can be seen by the following sup-chapters on how to start a master and worker, it is simple and has a low technical skill level required.

The framework with the custom implementation can be found at https://github.com/Andrioden/plugnhost/archive/v0.1.zip.

10.1 Starting the master

The master can currently only be run on a unix based operative system, as Nginx's load balancer module does not work on Windows. Note that the framework's core functionality is fully cross-platform. The master has only been tested on Ubuntu 12. The following are the steps to start the master node:

Step 1) Download the zip file.

Step 2) Install dependencies: Python v2.7

Step 3) Run the start script: "sudo python go.py master"

10.2 Starting the worker

Can be run on Windows and unix based operative systems. The worker have been tested on Windows 7 32Bit, Windows 7 64Bit and Ubuntu 12. The following steps to start the worker node are:

Step 1) Download the zip file.

Step 2) Install dependencies: Python v2.7

Step 3) Open the twisted communication port on the local router, and route it to the worker node.

Step 4) Run the start script: "sudo python go.py workfor [DOMAIN]" where DOMAIN is the url to the server you want to work for.

Step 5) Open the assigned http service port on the local router, and route it to the worker node.

11 Future Work

The framework is only a proof of concept, and to be considered a framework that can be used in a production there are several things that needs to be implemented and researched.

11.1 Functional improvements

Here is a list of functionality that would be an improvement to the framework.

- Detect worker nodes that are either completely failing or underperforming so it can be reported to the master service class.
- Allow services to be deleted and updated live, without requiring a process restart.
- Allow the developer to implement custom events in the master and worker service pair classes.

- The problem of distributing a database is a well-researched topic, and this project does not delve into that topic. However, it would be an important point in validating the usefulness of the framework to do a custom implementation of a distributed database with the framework. Together with a distributed web page it would be a more realistic scenario which could be applicable to real world problems.
- Can limit the amount of computer resources the server will use, the relevant resources are storage, RAM and CPU.
- Worker performance should be monitored and logged, this can be used to determine the value the worker node adds as well as give feedback to users on their contribution.
- Use the UPnP protocol to automatically open and route port traffic to the worker node, this would eliminate step 3 and 5 of the worker node starting steps.

11.2 Research topics

During this project a few related research topics have been discovered that has not been answered because they were either outside the scope of the project or was considered too large of a topic to be spent time on and included in this paper.

- In chapter [8.1] the problem of keeping control of a system using a p2p architecture is discussed, and it is concluded that a master-worker architecture is most likely the only possibility. However, this could pose an interesting research question. Is it possible to maintain control over a p2p system without *significant or total autonomy from central servers*?
- RQ4.1 was not answered in this project and is therefore listed under future work, the question is "*can it be avoided that an evil worker does permanent damage to the persistent storage?"* To further elaborate on this, in a scenario where worker nodes have write access to the distributed database, is it possible to safeguard the whole system from being permanently damaged by one evil worker. Is the answer simply backups? Is there ways to detect destructive patterns that could detect or suggest that a worker might be evil?
- RQ4.2 was also not answered in this project. The question is "*can sensitive data be stored at untrusted worker nodes?"* One example of sensitive data is user passwords. Even if they are hashed by a strong cryptographic algorithm they are still open for attack. However, un-hashed business data can also be considered sensitive. So the question asks if this data can be protected from being snooped on by an evil worker.
- Examine how patching of service software should be conducted on worker nodes that the developer do not have administrative access to.

References

[1] Basili, Victor R. The experimental paradigm in software engineering. Springer Berlin Heidelberg; 1993.

[2] Git. Distributed is the new centralized [Internet]; 2013 Jun. Available from: http://git-scm.com/.

[3] Wikipedia contributors. Value-added service [Internet]. Wikipedia, The Free Encyclopedia; 2013 May 21 [cited 2013 Jun 18]. Available from: http://en.wikipedia.org/w/index.php?title=Value-added_service&oldid=556124719.

[4] Dropbox. Dropbox - Plans [Internet]. Dropbox Inc; 2013 Jun. Available from: https://www.dropbox.com/pricing.

[5] Alexa. Wikipedia.org Site Info [Internet]. Alexa Internet, Inc; 2013 Jun. Available from: http://www.alexa.com/siteinfo/wikipedia.org.

[6] John Gaudiosi. Riot Games' League Of Legends Officially Becomes Most Played PC Game In The World [Internet]. Forbes. 2012 Jul 11 [cited 2013 Jun 18]. Players: [about 3 screens]. Available from: http://www.forbes.com/sites/johngaudiosi/2012/07/11/riot-games-league-of-legends-officially-becomes-most-played-pc-game-in-the-world/

[7] Steve Peterson. League of Legends and Riot's Play for Global Domination [Internet]. GamesIndustry International. 2012 Oct 17 [cited 2013 Jun 18]. Players: [about 1 screen]. Available from: http://www.gamesindustry.biz/articles/2012-10-16-riot-president-make-the-s-in-riot-games-mean-something

[8] Telenor. Vilkår for bredbånd over DSL [Internet]. Telenor ASA; 2013 Jun. Available from: http://www.telenor.no/privat/abonnementsvilkar/vilkaardsl.jsp

[9] Oram, Andy. Peer-to-peer: harnessing the benefits of a disruptive technology. O'Reilly Media, Inc; 2001. 16 p.

[10] Oram, Andy. Peer-to-peer: harnessing the benefits of a disruptive technology. O'Reilly Media, Inc; 2001. 22 p.

[11] Shao, Gary. Adaptive scheduling of master/worker applications on distributed computational resources [PhD diss]. University of California; 2001. 3 p.

[12] Milojičić, Dejan S., Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. ACM Computing Surveys (CSUR). 2000;32(3):241-299. 244 p.

[13] Milojičić, Dejan S., Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. ACM Computing Surveys (CSUR). 2000;32(3):241-299. 248 p.

[14] Dilley, John, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally distributed content delivery. Internet Computing, IEEE. 2002;6(5):50-58. p 53.

[15] Bryhni, H., Klovning, E., & Kure, O. A comparison of load balancing techniques for scalable web servers. Network, IEEE. 2000;14(4):58-64. p 58.

[16] Nginx [Internet]; 2013. Available from: http://nginx.org/en/.

[17] Nginx. Module ngx_http_upstream_module [Internet]; 2013. Available from: http://nginx.org/en/docs/http/ngx_http_upstream_module.html.

[18] Sam Kleinman. Use Nginx for Proxy Services and Software Load Balancing [Internet]. Linode Library. 2010 May 11 [updated 2011 Apr 29, cited 2013 Jun 18]. Available from: https://library.linode.com/web-servers/nginx/configuration/front-end-proxy-and-software-load-balancing#sph_front-end-proxy-services-with-nginx.

[19] anthonysomerset. Adding websites to nginx without restarting nginx [Internet]. Stack Exchange, Inc; 2011 Aug 21 [cited 2013 Jun 18]. Available from: http://serverfault.com/questions/303568/adding-websites-to-nginx-without-restarting-nginx.

[20] HAProxy. The Reliable, High Performance TCP/HTTP Load Balancer [Internet]; 2013 Jun. Available from: http://haproxy.1wt.eu/.

[21] Sang-Min Park. Preliminary benchmark for ELB [Internet]. GitHub, Inc; 2013 Feb 17 [cited 2013 Jun 18]. Available from: https://github.com/eucalyptus/architecture/blob/master/features/elb/3.3/elb-benchmark.wiki.

[22] ZooKeeper. ZooKeeper: A Distributed Coordination Service for Distributed Applications [Internet]. The Apache Software Foundation; 2013 Jun. Available from: http://zookeeper.apache.org/doc/trunk/zookeeperOver.html.

[23] Google. Developer Guide - Protocol Buffers -- Google Developers [Internet]. Google Inc; 2012 Apr 2 [cited 2013 Jun 18]. Available from: https://developers.google.com/protocol-buffers/docs/overview.

[24] MongoDB. Agile and Scalable [Internet]. 10gen, Inc; 2013 Jun. Available from: http://www.mongodb.org/.

[25] Twisted [Internet]. Twisted Matrix Labs; 2013 Jun. Available from: http://twistedmatrix.com/trac/.

[26] Node.js [Internet]. Joyent, Inc; 2013 Jun. Available from: http://nodejs.org/.

[27] Wikipedia contributors. WebSocket [Internet]. Wikipedia, The Free Encyclopedia; 2013 Jun 10 [cited 2013 Jun 18]. Available from: http://en.wikipedia.org/w/index.php?title=WebSocket&oldid=559267872.

[28] BOINC. Open-source software for volunteer computing and grid computing [Internet]. University of California; 2013 Jun. Available from: http://boinc.berkeley.edu/.

[29] Gray, J. Distributed computing economics. Queue; 2008;6(3):63-68. p 66.

[30] BOINC. BoincIntro - BOINC [Internet]. University of California; 2011 Des [cited 2013 Jun 18]. Available from: http://boinc.berkeley.edu/trac/wiki/BoincIntro.

[31] Superdonate. Donate Idle Computer Time to your Favorite Charity [Internet].

SuperDonate, Inc; 2013. Available from: http://www.superdonate.org/.

[32] Amazon. Amazon Elastic Compute Cloud (Amazon EC2), Cloud Computing Servers [Internet]. Amazon Web Services, Inc; 2013 Jun. Available from: http://aws.amazon.com/ec2/.

[33] Google. What Is Google App Engine? - Google App Engine [Internet]. Google Inc; 2013 Jun 11 [cited 2013 Jun 18]. Page views: [about 1 screen]. Available from: https://developers.google.com/appengine/docs/whatisgoogleappengine.