KTH Royal Institute of Technology

Highly Available and Robust Network Services in Under-served Areas

Jiannan Guo

Master's thesis

Supervisor:	Prof. Björn Pehrson, KTH
Examiner:	Prof. Markus Hidell, KTH

Stockholm, 15th September, 2014

Part I

Robust Network Infrastructure in Rural Areas of Tanzania

Part II

Highly Available Web Services

Chapter 1

Introduction

1.1 Background

Affordable, yet stable web services are highly desired in rural area, as a mean to alleviate digital divide and improve life quality. When it comes to under-developed regions in Africa, requirements and conditions need to be carefully assessed and analyzed, for that challenges could be unique and dramatically different than metropolitan.

1.2 E-learning for Open University of Tanzania

Open University of Tanzania (OUT) [1] is the first university of East Africa Region to provide open and distance learning programmes. To distribute course content through the whole country, Moodle has been chosen as underlying digital resource management platform. Moodle[2] is an open-source industrial-level online learning platform and resource management system. As a typical data-driven web service, Moodle runs over an underlying database and assemble its webpages on-the-fly based on user requests. It is written in PHP and heavily tested against Apache, Nginx and MySQL. At present, the platform is running as a standalone web service in a central server and mainly serve static content such as PDF, Text and Slides. Although OUT has the vision to introduce multi-media materials to enhance education quality. OUT also establishes learning centers in major cities and towns all over the whole country and is ambitious to extend to a larger scale. An emerging obstacle is to provide services in remote areas with poor network connection and bandwidth.

1.3 Problem Identification

As part of the project, we investigated local conditions and needs within the scale of Serengeti Broadband Network, especially in areas with evident

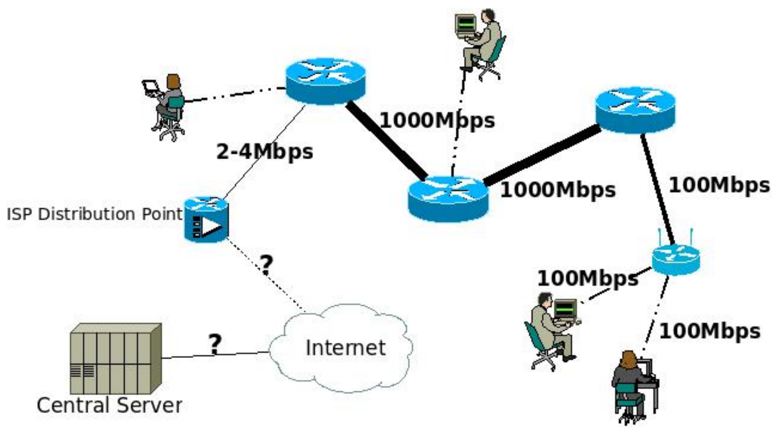


Figure 1.1: A Typical Setting of Rural Local Access Network (LAN)

demands of services and lack of infrastructures. And we were able to identify following challenges:

1.3.1 Power Outages

Power grid in rural areas of Tanzania is so unreliable that UPS for critical device is almost a must. While people are gradually adapting to mobile platforms, such as smart phones and tablets, backbone infrastructures are also required to be more persistent. Equipments powered up by solar and battery are highly desired due to cost-efficiency. Although power consumption need to be optimized in this circumstance in order to prolong battery life and improve reliability.

1.3.2 Poor network quality and frequent failure

Although local network is operated by ICT4RD project and can be fairly reliable, uplink is still depending on national-wide ISP and is somewhat unpredictable according to our observation. Network failure could occur anytime and can last for random period (several minutes to several days). Those web services that depend on a central server are apparently not accessible during the failure. On the other hand, the uplink can be very narrow due to poor infrastructure and limited budget. It could be difficult to squeeze multimedia services into such bandwidth.

To better illustrate this problem, suppose a typical setting in Figure 1.1. Major backbone components in this LAN are interconnected through fiber-optical lines, and network is distributed to users through WiFi or Ethernet. The LAN is linked to the Internet through ISP distribution link and central server resides on the otherside of the Internet.

Due to limited budget and ISP capacity, the upper link is equipped with an average bandwidth of 2 4Mbps which is shared among all users in

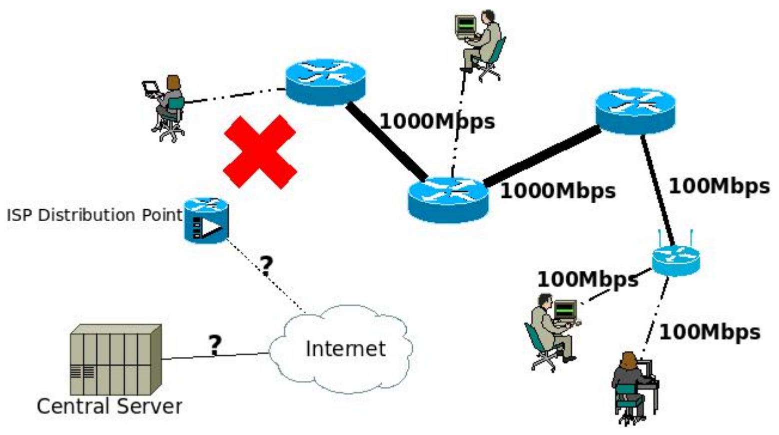


Figure 1.2: Uplink Failure leads to the isolation of LAN

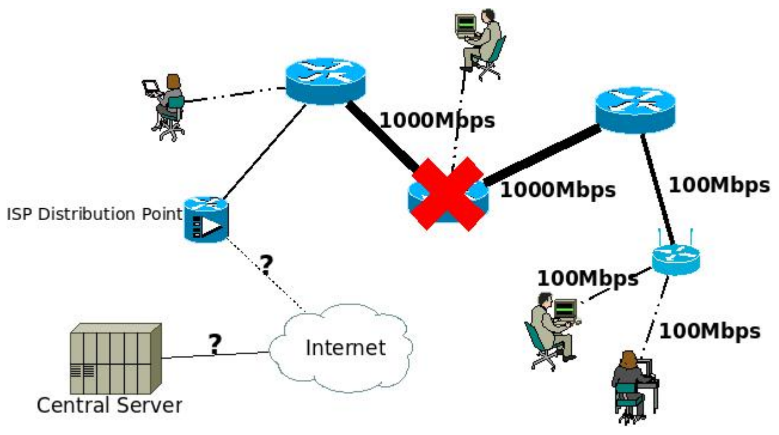


Figure 1.3: Network Separation due to Component Failure

LAN. While a minimum bandwidth of 1.5Mbps is recommended for video streaming, it is difficult for users to get decent service from central server. To worsen the situation, the upper link is somewhat unpreditable, which leads to the isolation of LAN, as shown in 1.2.

On the other hand, components in the LAN can also break down which leads to network separation, see Figure 1.3. In the first case, multi-media content can hardly reach end users. And in other two cases, users cannot get service at all.

1.3.3 Limited budget

Cost is an essential factor during rural ICT development. Given relatively smaller user base and weaker demand, equipments need to be chosen wisely. Although future maintainence and development also need to be considered.

Chapter 2

Adapt to Frequent Network Failure and Limited Bandwidth

2.1 A closer look at the problem

As introduced in section 1.2, Moodle is deployed as underlying course management system for OUT E-learning platform. Moodle is an open sourse project written in PHP and well-documented[3][4]. Similiar to other web applications, it can be deployed in a typical LAMP or LNMP stack. In this chapter, we mainly focus on possible solutions for two problems stated previously, and leave the choice of actual server to chapter 3

Moodle is a typical database-driven web application where all the pages are generated on-the-fly based on user request. The whole application is composed of three main components:

- PHP source code, typically in `/var/www/moodle/`
- A database to store data or metadata including site configuration, student information, course details, events, etc.
- A directory to store materials and resources, as well as cache and temparory files. Typically it is named as `moodledata/`

The problem addressed previously can be simplified and modelised as following, see Figure 2.1. Each node in the model denotes a local server/proxy and has a certain amount of users associated with it.

As simple as the model might be, components in it could be vastly heterogenous while mapping to different techniques. Content stored in a node can either be web objects, SQL replies, codes or even entire databases. Communication in between can also be based on a variety of protocols.

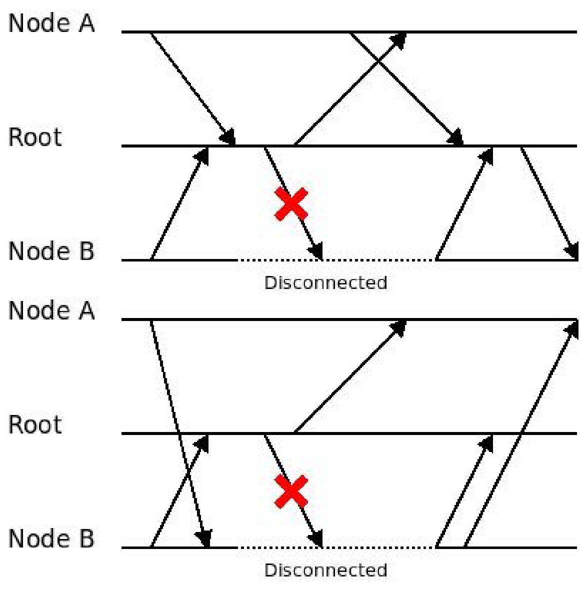


Figure 2.1: Web Delivery Model

As an online learning platform, users do not only passivly accept information, but also interact with Moodle through forum, personal blogs and quiz. All the changes made by users must be stored and seen everywhere. Thus, the system should not be read-only under any circumstance.

Moodle has been in service and adding new services should affect existing structure as less as possible. Also, steps of adapting changes should be properly designed to avoid crushing the service.

To maintain consistency and serve up-to-date content, a reasonable amount of communication overhead is necessary and is normally positive proportional to the extent of consistency. Although, due to the presumption of poor network connection and narrow bandwidth, different nodes in the system are preferably decoupled and autonomous.

The autonomy is also closely correlated to the ability of performing offline operation. Many distributed systems have the ability to detect and recover from network partitioning, although it normally leads to a compromise of consistensy and content freshness. When a user request a page, Moodle loads all previliges of the user, generate pages accordingly and log the session. This results in uncacheble content and interaction-must logins. It has been proven that consistency, high availability and partition tolerence are impossible to be achieved at same time[5][6], necessary assumptions must be made according to the condition and needs.

While the majority of web caching and content distribution techniques aim at better performance and delivery efficiency, we prioritize the ability of performing basic functionalities during network failure. We tolerate a relatively loose consistency while ensuring eventual convergence.

Lastly, to realize affordability, we mainly focus on open source techniques and free ware. Thankfully, many successful projects and tools have been made open source and publicly available. In the following sections of this chapter, we evaluate a variety of techniques against the criteria stated above and propose our solution based upon the conclusion. Several of potential solution are also tried out.

2.2 Survey on Techniques to enhance web performance

2.2.1 Content Delivery Network

Content Delivery Network overlaps with Web Cache Proxy at the concept of pushing web content to users. A Content Delivery Network is a collaborative set of surrogate servers spanning the network, where web contents are mirrored[7]. Users will perceive a smaller latency while fetching content from a nearby CDN surrogate server rather than original web server. The essence of CDN is illustrated in figure X

Since more and more web services are evolving to provide dynamic content, CDN also takes advantages of cachebility hints when dealing with dynamic contents[8].

2.2.2 Simple Web Caching

An intuitive and common solution for the problem of limited bandwidth is to cache popular web content locally, as illustrated in Figure 2.2. A server-side web cache proxy typically sits in front of web server, attempting to serve user request with cached objects rather than triggering computational workload on web application. Web caching has been proven to be an effective approach to reduce bandwidth usage, user-perceived latency and loads on original server[9].

Web cache is greatly advantageous in our scenario that it does not require modification on web application, except that some TCP optimizations could be done between web cache proxy and web server frontend. Web cache proxy can continue serving requests if offline mode enabled, which also meets our requirements. The traffic between cache proxy and authentic server is standard HTTP request and response. The overhead and frequency of communication mainly depends on cache hit / cache miss ratio, expiration time and cache directory size.

Although, all write traffic traverses cache proxy and goes to original server, which leads to a read-only system immediately when network disconnects. This is magnified when Moodle is taken into consideration, where user logins require interaction with original PHP code and database.

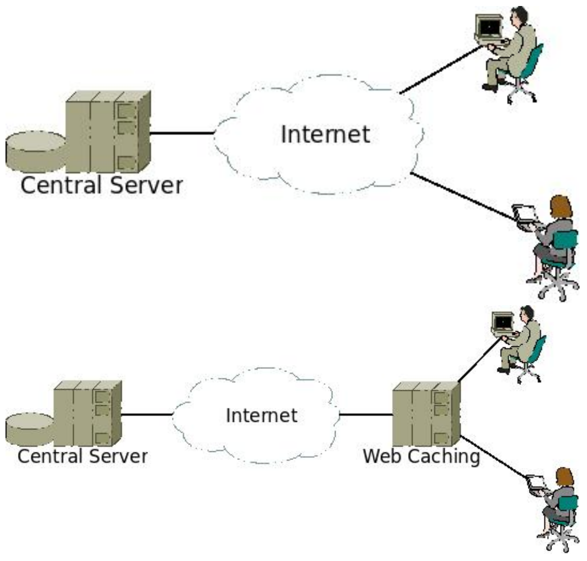


Figure 2.2: Serve users without and with a Gateway Cache

2.2.3 Web Caching with Edge Server

To address the issue of dynamic content generation and client-server interaction, an intuitive and brute-force solution is to generate user-specific page at the edge. A comprehensive study of edge servers can be found in **??**. Four strategies are presented: (a) edge computing (b) content-aware caching (c) content-blind caching (d) data replication, see Figure X In each of the strategy, the edge server attempts to reply user request on the behalf of original server with the information that is locally available. Edge computing still heavily relies on central database, hence out of our consideration. While CAC and CBC store partial database at the edge, data replication stores a complete copy of the database. When the size and complexity of database permit, data replication is desired since it outperforms other strategies in both response speed and offline operation. Although, it adds another layer of complexity to perform transaction processing and maintain the consistency through mutliple distributed databases. An exhaustive survey is presented in section 2.3.

Application code rarely changes in our case and is always on-way synchronized from original server, thus consistency can be relatively easy to achieve by periodically utilizing tools like rsync**??**.

2.3 Multi-Master Database Synchronization

Database replication techniques has been intensively deployed over clusters and workstations for redundency and load balancing (Figure X). A

LAN is always assumed by most of database replication techniques, whereas database replicas across the WAN are desired to realize our goals. We prioritize availability by compromising on consistency, as long as the system can reach eventual consistency**??**. Offline operations on distributed databases implies simutaneous modification on the same entry, which result in conflict after reconnection. Techniques to detect and reconsile conflicts are studied and evaluated.

2.3.1 Database Cluster

Firstly, several databases with native support for replication are studied.

CouchDB ?? is an open source distributed database system developed in Apache. The most attractive feature of CouchDB is that it natively supports bi-directional synchronization among multiple database replicas and offline operations. When one replica is disconnected from the network, it retains autonomy and continues as a fully functional database from user point of view. Although, it fails to be our candidate since it is NoSQL database, whereas Moodle heavily relies on SQL calls and it will a significant task to modify Moodle to use NoSQL database.

MySQL Cluster ?? is an open source distributed database system based on MySQL. It supports database sharding and duplicating. A typical use case of MySQL Cluster is shown in Figure X. Although, redundent copy of database can only be accessed with the presence of management master, and cannot be updated during network failures. Furthermore, database nodes are closely coupled with the assumption of LAN (low latency and high bandwidth).

2.3.2 Middleware-based Repliation System

Several studies also proposed the approaches to solve database synchronization at middleware-level**?? ?? ??**. C-JDBC **??** is an Java implementation of RAIDb**??**, aiming at a framework to manage heterogenous databases. With built-in functionality of scheduling transaction processes, C-JDBC is perceived by users as a single virtual database. Although, the system is still centrally managed and could not handle partitioned network. Ganymed middleware system**??**, inspired by C-JDBC, achieves consistency by serializing update/write requests at master and propogating changes to replicas in a lazy fashion. Users see a consistent data state (snapshot isolation), even though stale might it be. The limitation of these two middleware is also clear, that no write can be served during network failures.

2.3.3 Multi-Master Sycnhronization System

Similiar to MySQL Cluster Multi-Master setup, we found three state-of-the-art open source tools to the similiar end.

- **Galera**[10] is an open source synchronous database replication software developed to scale web application and provision high availability. It achieves consistency by optimistic locks and group communication. Galera is quorum-based and handles network partitioning by sacrificing the minority. Hence, Galera lacks the essencial features that we are seeking for and is out of consideration.
- **Tungsten**[11] replicates database asynchronously and allows loose consistency. Transactions are commited locally, and then propogated across all other nodes. Tungsten features offline operation and automatical recovery although it leave the responsibility of conflict avoidance completely to the application. Conflicts may disturbe replication and hard to trace.
- **SymmetricDS**[12] is another open source asynchronous database replication management tool. It has several built-in rules to detect and reconsile conflicts. It can be configured flexibly to meet different needs, for example, have different courses store in different sites. Also, one appealing feature is that SymmetricDS support file synchronization as well. Figure X birefly illustrates how SymmetricDS works.

We can draw the conclusion from above comparison that SymmetricDS is closest to our critaria and can be properly extended to meet our requirements. In the following section, we explore SymmetricDS in details and propose an approach to extend it.

2.3.4 Operational Transformation

Chapter 3

Low Power, yet Powerful

The ARM-based processors have received great attention for its characteristic of low power consumption and energy efficiency, especially in smart phone and portable device industry, where power consumption is one the most critical specifications. Furthermore, there is trend in server industry to shift to ARM-based architecture in order to cut off the bill of electricity. ARM is also ambitious in this area and about to publicize processors capable of virtualization. When it comes to rural development, power shortage has been forcing researchers and engineers to seek for alternative power sources. Lower power consumption of the equipments implies a bigger potential of surviving severe environment. Currently, we are exploring the possibilities of two platforms, namely Raspberry Pi and Odroid. The specifications of these two platform can be found in Table X In the following sections, we benchmark a variety of attributes of Odroid and Raspberry Pi. The objective is to clarify the capacity of these two platforms and an optimum form to run the web service. We test every component of a complete Moodle installation to identify the bottleneck of the application. Based on the results, we propose the optimum cluster solution that can be easily scaled out to serve more users.

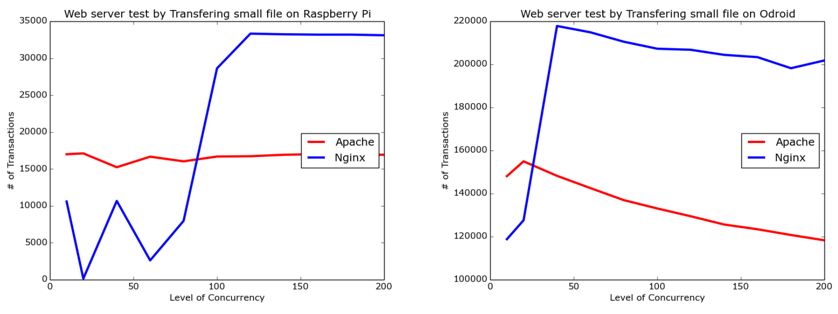
3.1 Processor and Bandwidth Latency

3.2 Benchmark of Web Components

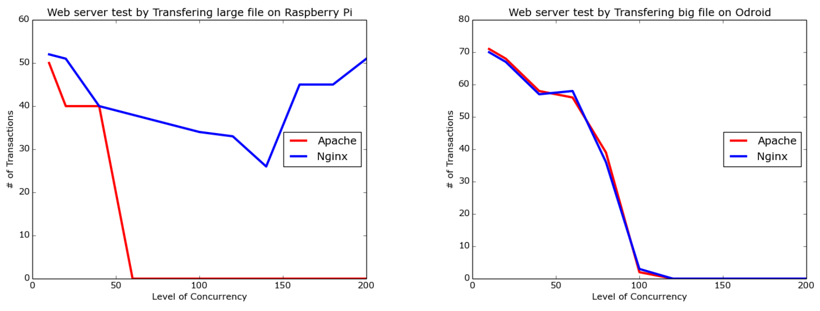
To test components of a complete Moodle installation, we design a expirement in Figure X. Each part is respectively substituted with either Odroid or Raspberry Pi, and remaining parts are running on a high-end machine which is much more powerful than these two platforms, in order to put enough siege on the testing target. Furthermore, simulated requests are generated from high-end machine as well.

3.3 Web Frontend

Most of the websites today are powered by Apache due to its long history and abundant extensions. Although Apache relies on a processed-based manner to handle new connection, which has limited the scalability and concurrency. Nginx is an event-based reverse proxy that handles request asynchronizily. It addresses C10K problem from the beginning and focus on scalability. To determine which server runs better on a resource constraint platform, we benchmark Nginx and Apache on both platform to evaluate the throughput, level of concurrency and response delay. We use siege to generate workload and compare the performance of web server with two different type of object: small text file and large JPG file. We siege the server in following setting:



(a) Raspberry Pi (small text file) (b) Odroid (small text file)



(c) Raspberry Pi (big jpg file) (d) Odroid (big jpg file)

Figure 3.1: A Benchmark of static file request on both platforms

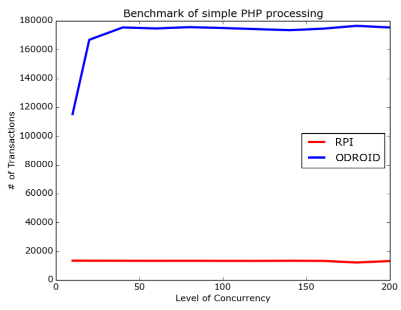
As shown if Figure 3.1, Nginx and Apache achieve comparable performance under a low level of concurrency, although Nginx outperforms Apache notebly when concurrency level increases. To be noticed, both web servers perform indistinguishably while serving large file. We observe fully occupied CPU in this specific test, which is due to intensive OS kernel processing of socket manipulation and packet tranfer. The impact of web server on the CPU is neglectable in this circumstance. Thus, expirement result in Figure

3.1(d) does not denote same performance of web servers.

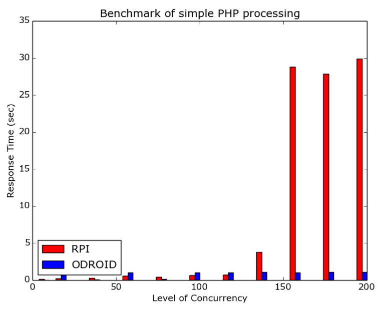
3.4 PHP Processing

As a large PHP application, Moodle requires significant computational resources for PHP processing. Thus, a testbed is formed to benchmark PHP processing capacity of two platforms, see Figure X The inputs are two different PHP scripts:

- a php script simply echoes `Hello, world!`
- Moodle index page which requires heavy php processing.



(a) $\sum trans$



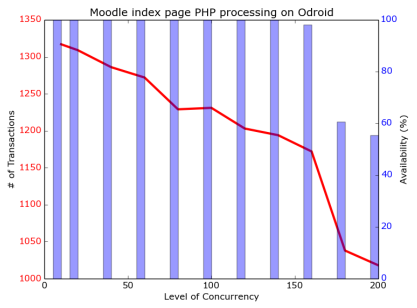
(b) Longest Response Time

Figure 3.2: A Benchmark of simple PHP processing on both platforms

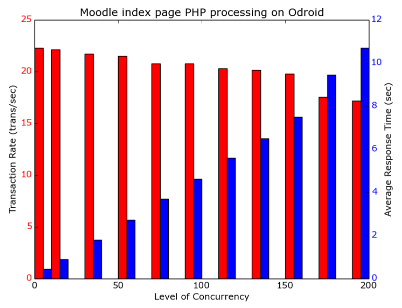
Figure 3.2(a) shows that Odroid can handle much more PHP request than Raspberry Pi. Overall response time is shorter for Odroid. Although, half of the CPU usage is taken by OS kernel during these two test and the difference satisfactorily convincing. The gap between two platforms is more evident when tested with heavier PHP processing, as shown in Figure 3.3.[1]

As shown in Figure3.3(a), transaction rate and availability drops under 180 concurrent requests occur. The latency is beyond tolerable under 50 concurrent requests, according to a study of tolerable waiting time of website**??**. As the total transaction number in this test is significantly lower than the previous one in Figure 3.2, we suspect that PHP processing power is the bottleneck in a moodle installation rather than database query and web frontend. Thus, we test the capacity of a standalone installation of Moodle, in which all components are in one box (Odroid), shown in Figure 3.4.

[1]We observe a 6 7 seconds delay to process index page of Moodle on Raspberry Pi and it can barely tolerate the mildest siege. Thus, the test result of Raspberry Pi is left out in this test.

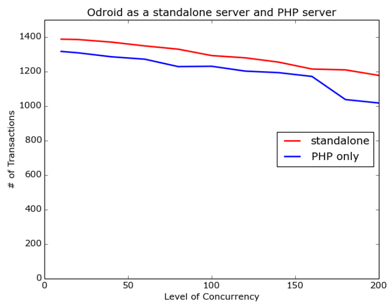


(a) $\sum trans$ & Availability

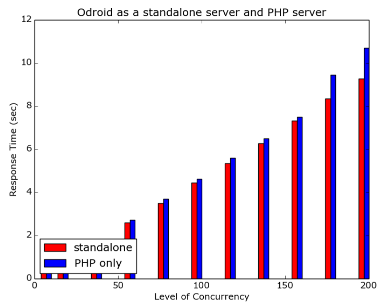


(b) $Rate_{trans}$ & T_{resp}

Figure 3.3: Moodle index processing on Odroid



(a) $\sum trans$ & Availability



(b) T_{resp}

Figure 3.4: Odroid as a standalone server and PHP server

Impact of adding web frontend does not evidently influence transaction rate, which implies that PHP processing capacity is the limiting factor in one Moodle installation.

3.5 Database

We apply the same method to test database although fail to exhaust the resource on Odroid before running out CPU and memory on our testbed high-end machine, hence we conclud that database query does not limit a Moodle installation before expanding PHP capacity.

3.6 Scale Out to Eliminate Bottleneck

To benefit most from currently available hardwares, we propose to form a small scale cluster than can be easily scaled out to cope with an increase of users. Furthermore, we investigate multiple different formations to find out the most economical setting. An intuitive solution is to put different services in physically separated hardwares, as a structure shown in Figure 3.5. Nginx server in Rapsberry Pi features both web server and load balancer with upstream PHP servers running in Odroid. To cope with the increase of users, this setting can be easily scaled out by simply adding more hardware for PHP processing. Although, as we compare the Siege test result with a

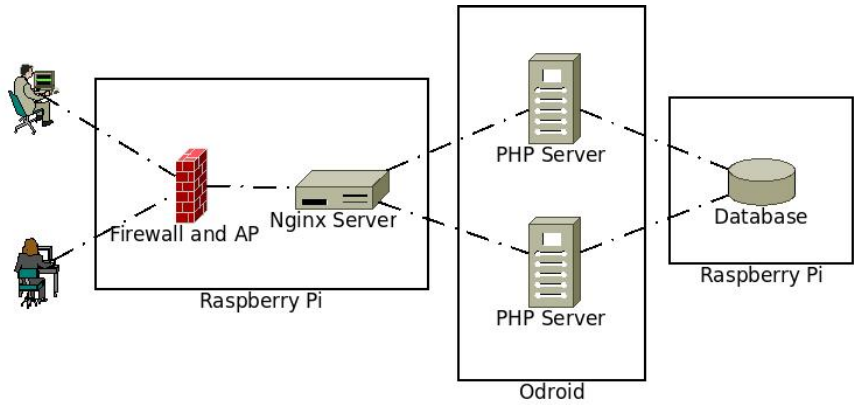


Figure 3.5: A naive form of cluster

standalone installation where all services run in one Odroid, we encounter a slightly lower performance, which is even worsen after we add one more PHP server, see Figure 3.6.

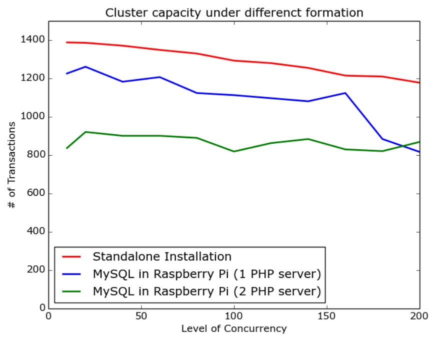
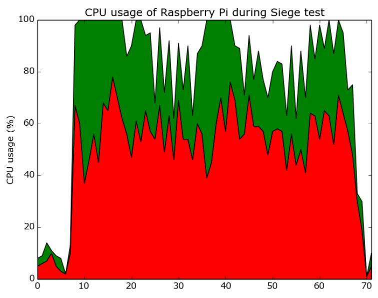
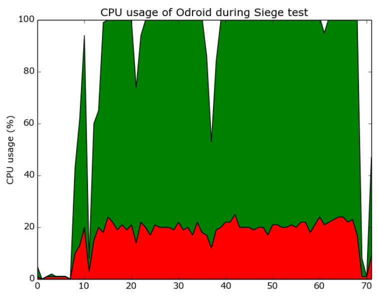


Figure 3.6: The Performance of running Database in Raspberry Pi

To further identify the cause, we siege the cluster with 100 simulated concurrent clietns and observe CPU usage of servers (CPU of Nginx server is far from full load hence left out in the result), the result is shown in Figure 3.7. We deduce that most of CPU resource in database server is consumed by OS kernel to handle concurrent sockets. Hence, database needs to be more closely coupled with PHP server, which results in the cluster formation in Figure 3.8. And the result meets our assumption that server capacity improves by adding extra PHP server, see Figure 3.9



(a) database



(b) PHP

Figure 3.7: CPU usage of servers

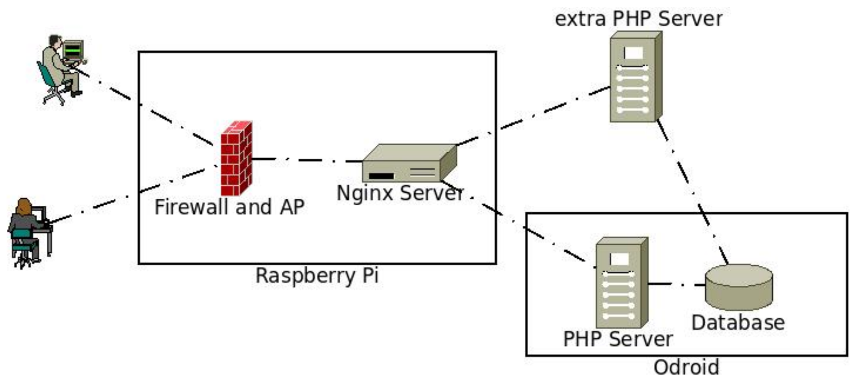


Figure 3.8: A better form of cluster

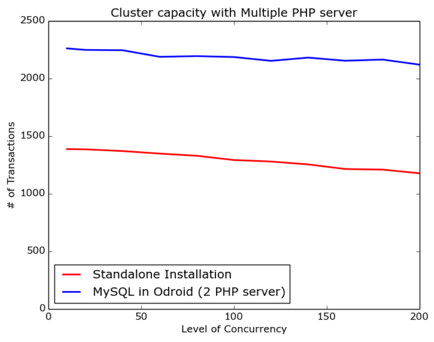


Figure 3.9: Performance of cluster with multiple PHP servers

To further boost web performance, we configure Nginx to cache frequently requested page hence reducing the load on PHP and database server.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

4.2 Future Work

Bibliography

[1] http://www.out.ac.tz/. [Online]. Available: http://www.out.ac.tz/

[2] https://moodle.org/. [Online]. Available: https://moodle.org/

[3] http://www.aosabook.org/en/moodle.html. [Online]. Available: http://www.aosabook.org/en/moodle.html

[4] https://docs.moodle.org/. [Online]. Available: https://docs.moodle.org

[5] E. A. Brewer, "Towards robust distributed systems," in *PODC*, 2000, p. 7.

[6] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[7] M. Pathan, R. Buyya, and A. Vakali, "Content delivery networks: State of the art, insights, and imperatives," in *Content Delivery Networks*. Springer, 2008, pp. 3–32.

[8] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *Internet Computing, IEEE*, vol. 6, no. 5, pp. 50–58, 2002.

[9] B. D. Davison, "A web caching primer," *Internet Computing, IEEE*, vol. 5, no. 4, pp. 38–45, 2001.

[10] http://galeracluster.com/. [Online]. Available: http://galeracluster.com/

[11] http://www.continuent.com/solutions/clustering. [Online]. Available: http://www.continuent.com/solutions/clustering

[12] http://www.symmetricds.org/. [Online]. Available: http://www.symmetricds.org/