Participatory Cloud Computing: The Community Cloud Management Protocol

Taariq Mullins

A thesis presented for the degree of
Masters of Science

ISAT Laboratory
Department of Computer Science
University of Cape Town
South Africa

August 2014

Dedicated to

My mother

Participatory cloud computing: The community cloud management protocol

Taariq Mullins

Submitted for the degree of Masters of Science
August 2014

Abstract

This thesis work takes an investigative approach into developing a middleware solution for managing services in a community cloud computing infrastructure predominantly made of interconnected low power wireless devices. The thesis extends itself slightly outside of this acute framing to ensure heterogeneity is accounted for. The developed framework, in its draft implementation, provides networks with value added functionality in a way which minimally impacts nodes on the network. Two sub-protocols are developed and successfully implemented in order to achieve efficient discovery and allocation of the community cloud resources. First results are promising as the systems developed show both low resource consumption in its application, but also the ability to effectively transfer services through the network while evenly distributing load amongst computing resources on the network.

Declaration

The work in this thesis is based on research carried out at the ISAT Laboratory, the Department of Computer Sciences, The University of Cape Town. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text. I hereby declare that this written work I have submitted is original work which I alone have authored and which is written in my own words. With the signature I declare that I have being informed regarding normal academic citation rules and I conform to citation conventions customary to the sciences. This written work may be tested electronically for plagiarism.

Taariq Mullins	Date

Acknowledgements

There are many people to thank for me being here. I am very grateful to have been awarded this opportunity to dedicate myself and my time to a task that will hopefully benefit others. The University, UCT, and the National Research Foundation have to be thanked for making this possible. But before all of that, my Supervisor, Dr Bigomokero Antoine Bagula, has to be thanked for his patience and time. All the afternoons we spent in his office discussing the craziest ideas relating to our work only served to expand my consciousness in a way I never imagined. His impressive work and dedication to his students will be missed by the University and the students. My friends and family for motivating and supporting me along the way. Devin, Chris and all the staff at WhereIsMyTransport for supporting me and providing me with the flexible working hours so I could finish my masters. Lastly, I have to thank the Daniels family. Most importantly, Nicole. You helped me out when I needed help the most, and did so in ways I could never repay. I will always love you and you are forever in my thoughts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Why Cloud Computing

As the division of labour increases in line with the automation of industries, human intervention in labour becomes more robotic[1]. The machine like operation of factory workers can be observed all around the world, where due to the division of labour it is easier for factory owners to subscribe machine intervention. But computer science is not only related to issues of production. It is involved in almost every aspect of being in modern society, from e-voting stations[2] to medical procedures which involve participants which are continents apart[3].

Cloud computing is one such method where we aim to make the role of computing in society more ubiquitous and alleviate the strain placed on organisations that require large amounts of infrastructure in order to execute their business requirements. The problem faced with carrying out such tasks is that for each business that has these requirements, all of them require massive expenditure for hardware and software solutions which do not always guarantee optimal solutions[4][5][6].

Problems arise from the fact that hardware is not always utilised to their maximum capabilities and service providers/businesses need to cater for their highest user requirements (an example is catering for largest volume in processing or bandwidth) resulting in underutilized hardware for the majority of the equipment lifetime. Indie developers and start-up companies are also not usually able to outlay the kind of capital that this sort of infrastructure requires[4].

Thus cloud computing came to be the saviour of enterprise in that third party providers could ensure a scalable distributed architecture would ensure that customers could ensure that their applications and services could be accessed world over and ensure uptime based on Service Level Agreements (SLA) with their third party solution providers. This platform is shared amongst other clients with similar requirements[7].

Why Participatory Cloud Computing

There is a compelling case for the cloud to be provisioned from a network topology such as that provided by grid computing[8]. This would allow spare resources on networked computing devices, not just personal computers but routers, supercomputers, storage systems, data sources, specialised devices, mobile phones to join the grid and perform actions to thus further enable the virtual data centre. Which, conventionally is a space used to solve specific problems in science, engineering and commerce[9].

The solution described here forms a so called community cloud. This is different from cluster computing in that members or participants in the grid, that form the cloud fabric, are not tightly coupled, heterogeneous platforms in terms of both hardware and operating systems as well as being more geographically distributed[8][9].

Digital Ecosystems are self-organising, scalable and sustainable distributed adaptive open socio technical systems inspired by natural ecosystems. This employs local players to play a pivotal role in our modern era of globalisation to locally create value networks at a global level[8]. Green Computing is the efficient use of computing resources with the primary objective being to account for the triple bottom line, i.e. people, planet and profit. This captures a broader means of measuring an organisation's success in terms of economic, ecological and social impact. These are values for measuring organisational and societal success[8]. Community cloud computing puts forward the proposition that to combine concepts from grid computing and autonomic computing, principles from digital ecosystems and sustainability from green

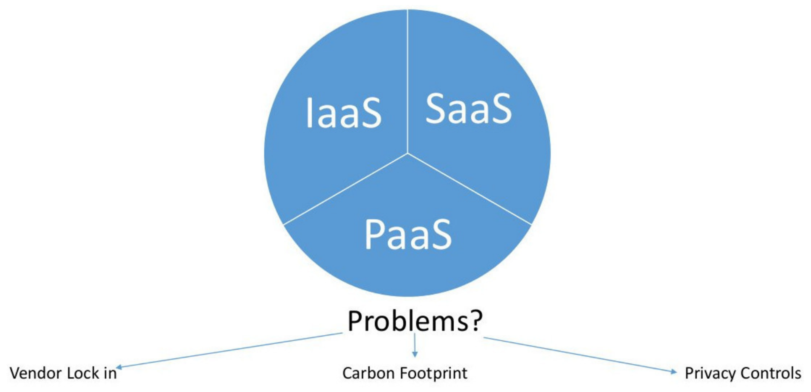


Figure 1.1: Three most common commercial cloud computing implementations: **I**nfrastructure **as a S**ervice, **S**oftware **as a S**ervice and **P**latform **as a S**ervice

computing to form the cloud would provide an alternative to the current implementation. As there are no concerns or flaws with current cloud conceptualisation that have been raised, problems lie with current vendor implementations[8].

Community clouds can be viewed as a step up or progression from academia-focused grid

Computing. This relates to cloud computing in that virtualised services are being provisioned over the internet without the users having knowledge, expertise or control of the infrastructure that is running the services they are requesting.[8] When using the word grid, further on in this thesis, we are referring more towards the physical and semantic nature of the network

It is also posited that community cloud computing is much more of a social infrastructure which implies ownership of all participants in the network of it[8]. The nodes are autonomous meaning users are still free to use their systems as they please, thus vendor control of such systems would be non-existent. A community cloud is not owned by any one particular vendor or organisations and therefore transcends the current generation of frameworks with regards to organisation lifespan which is directly related to product and support lifespan.

Artur Andrzejak displays how the desktop user would participate in this sort of scenario and can become a contributor in the face of large competing vendors such as Microsoft and Google [10]. Where the user's desktop machine joins the grid and is able to reclaim his or her workspace at any time, synonymous with Amazon's spot instances.

Community cloud computing is envisaged to have a lower carbon footprint than current data vendors[5]. This is based on the assumption that underutilised user machines consume less energy than dedicated data centres. Embedded devices, such as those powered by DC power sources (modems, routers), use the exact same amount of power when they are idle compared to when they are operational. (Obviously if the device has advanced power management capabilities this is not the case but this usually drives up the prices of electronics.) These sort of devices could also participate in such frameworks.

A community currency, free from a central authority such as a particular reserve bank or nation backing it, would be required to run and consume services in the cloud. Here service providers, large or small, would be compensated by a form of digital currency for hosting and providing consumed services to the cloud which could be used to consume other services. Community cloud computing sees this paradigm facilitating existing cloud computing vendors to gather a significant amount of this currency which they can monetise against participants running a community currency deficit[8].

The relevance of using wireless sensor technology in the developing world to enhance research in environment monitoring has been stressed in [11] and a ubiquitous sensor network architecture that reveals the importance of a middleware to hide the complexity of the lower layers of the architecture from the application layer was described in [12]. The focus of the work presented in this thesis was on lightweight processing to mitigate the resource limitations associated with lightweight devices such as sensor motes which are often deployed unattended with instable power supply. This work may be used to design intelligent middleware for wireless sensor networks when applied in the context of pollution monitoring [11], smart irrigation [13], water quality monitoring [14] and drought mitigation as proposed [15].

1.2 Related Work

The cloud is conceptually made up of three layers. These are the coordination layer, resource layer and service layer. Where the service layer is the tertiary layer and coordination is the primary layer.

It is proposed we borrow from economics the market oriented approach to dealing with commodities[6]. This follows from what was discussed earlier that if an open collaboration standard is established for members of the cloud, services can be commoditised and thus traded. A directory would be required for tertiary level providers to locate the appropriate primary level providers with the resources that they require in order to perform their desired tasks.

An auctioneer would be required to control requests from various participants for resources, suppliers and consumers. The auctioneer(s) would be an independent authority that does not represent the consumer or provider in order to guarantee impartiality but since they are in total control of the trading process they need to be trusted by the users[9].

Brokers would act as the middleman between the tertiary and primary levels in the same way a broker functions today via requests in the market directory. It will buy up resources from providers and sub-lease these to consumers on a needs basis. This way locating providers to provide geo-dependant services becomes more appropriate and easier to do. This also means that providers will be able to dynamically expand or resize their resources based on workload demands. Thereby enabling these providers to stay within the bounds of their SLA, by providing reliable location aware QoS aware services[9].

Optimum resource allocation will have to be calculated based on service requirements. This has to be maximised at all times due to rising costs of energy as well as operating costs of larger participants in the cloud. So it will be essential that service requirements are calculated along with resources available, taking into account the location to provide optimum QoS aware applications becomes quite a complex task for a constantly changing environment. This is a multi-dimensional optimisation problem which the solving of is tasked to heterogeneous optimisation algorithms such as hill climbing, dynamic programming, parallel swarm optimisation

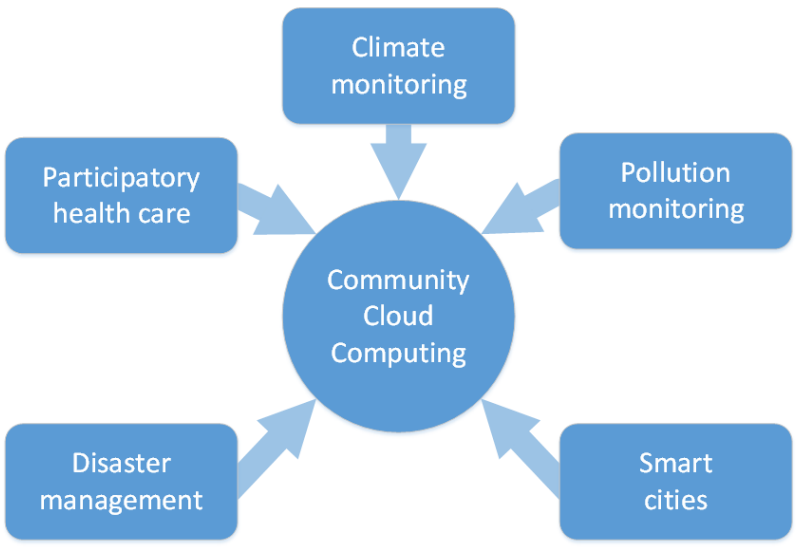


Figure 1.2: Various possible applications for a community cloud infrastructure

and multi-objective genetic algorithms[9].

Monitoring is essential to optimal functioning of the cloud. Via real-time sensor updates systems can be monitored and adjustments can be made, should critical levels be reached in terms of heat, power and performance. Nodes should also be allowed to fail gracefully allowing services to continue uninterrupted in the cloud. This information should be provided in a manner that is scalable throughout the cloud[9].

When this is coupled with protocols such as Yao's protocol, this allows data to be secured. It uses secure two party computation to obfuscate (encrypt) data and then de-obfuscate it on the client side to reveal the results of the computation[16]. This requires communication between the cloud and client during computation. By ensuring that facilities such as traceable access, rights management (adding and revoking privileges), fine granularity and security parameters are in place, we can provide data protection to information stored in the cloud. Decryption is done as late as possible and performed usually on the end user devices.

These end user systems could also include mobile phones, as this has proved to be a useful resource when coupled with medical data gathered in the field by the Fontane project[10]. This would allow doctors to monitor patients and get alerts regarding particular patient health status without having to participate in heavy computational queries when receiving and reviewing data[10].

1.3 Contribution

This thesis proposes to address some of the issues related to cloud computing by proposing a participatory cloud computing framework where a lightweight database management system is used to store data and two protocols are designed to achieve resource discovery and allocation. The developed framework, in its draft implementation, provides networks with value added functionality in a way which minimally impacts nodes on the network. Two sub-protocols are developed and successfully implemented in order to achieve efficient discovery and allocation of the community cloud resources. First results are promising as the systems developed show both low resource consumption in its application, but also the ability to effectively transfer services through the network while evenly distributing load amongst computing resources on the network.

The thesis extends itself out of a previous work conducted by the author for the honours thesis with the title, ”An Adaptive Middleware in a Participatory Cloud Computing Environment” in 2012. The investigation that was conducted in a similar fashion using the Tahoe-LAFS file system and SNMP to produce a middleware that would allow us to provision services in wireless mesh networks to provide cloud like services. The work was submitted as a paper at SAICSIT 2013[17]. Following this a paper was submitted to IEEE-UIC 2013[18] and was published in the subsequent conference proceedings.

1.4 Thesis outline

This thesis follows a somewhat unconventional flow. This is partly inspired by a submitted paper by the author which was presented at IEEE-UIC 2013. Each chapter presents itself as its own study into an acute area of work within the project; each containing its own abstract, introduction, readings, investigations and conclusions where appropriate. This will allow for much more fluid reading of the work as the examinations are all distinct from each other while each step being reliant on the prior work. The flow of the document is broken up as follows. This chapter includes an introduction into the work as well as providing an overall context within which the project was borne and framed.

Following this is a design chapter of sorts where the details of the framework or architecture are described herein. This will include descriptions of all the protocols developed. Thereafter 3 chapters will follow which provide both the rationale for decisions taken in selecting these designs as well as proving the solution. It must be noted that the solution provided or rather described in these 3 chapters represent a draft implementation of what is a much broader work. After these chapters the document is wrapped up in its concluding remarks and also presents the user with possible future projects that could enable further research efforts.

Chapter 2

Community Cloud Management Protocol

Abbreviations:

6LoWPAN: IPv6 over Low power Wireless Personal Area Networks

Swap space/area: A memory storage area on Linux akin to virtual memory on windows.

802.11b/g/n: A group of wireless protocol standards used world wide in wireless device communication

IO: Input/Output

AES: Advanced Encryption Standard

CBC: Cypher Block Chaining

2.1 CCMP

What is a cloud? The cloud commonly refers to a networked environment usually separate from the user's own immediate computing environment, where computing services are made available for the users with appropriate access.

What is a community cloud? This commonly refers to the practice of users volunteering their own private hosts for use in a grid computing fashion. Where each user that connects to the network, volunteers a few of their available resources on the network adding to the resource pool of the network. These networks usually span over much larger areas than conventional cloud infrastructures, which are usually confined to data centres. The resource pool can consist of either raw computing resources such as hard drive space and CPU time, or it can consist of services running on the host such as VoIP, printing or web services.

2.2 Project framing

To frame the project better we can consider an example of a wireless sensor network being deployed in a rural setting monitoring the moisture content in the soil. Typically these installations consist of a large number of small wireless sensor devices connected wirelessly to each other using a low power wireless protocol such as 6LoWPAN[19]. The wireless range of these devices typically extend to quite lengthy distances if outside and within line of sight of additional access points running the same protocol. However, this does not compare to the range one can usually achieve when connecting with network protocols such as 802.11b/g/n[20].

These wireless sensor nodes are usually connected to a gateway device which in turns connects with other gateway devices, communicating with 802.11b/g/n, to form a mesh network. At some predefined point in this network a termination node is selected. In this context the meaning of a termination node is one that acts as a gateway between the private wireless mesh network and a user operating outside this wireless environment. This termination node could also provide external services to nodes in the network such as internet access but that will not be expanded on further in this document.

In order for sensor data to be collected, each node would have to be visited by the administrator of the network. This does not lend itself to be very flexible, as nodes can go down and data can become unavailable. Additionally, if services were to be provisioned on the network, it would need to be done without overburdening one node in the network.

By aggregating these resources and providing a way of easily allocating services

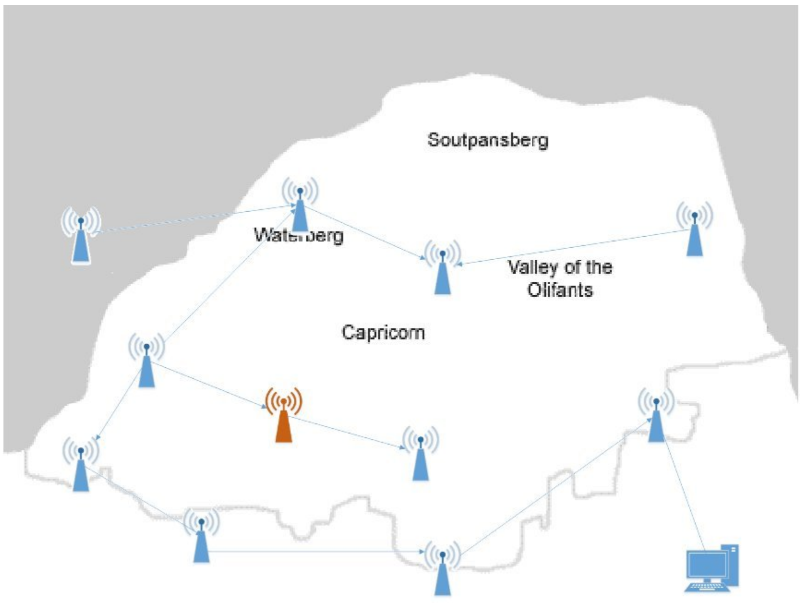


Figure 2.1: example of a long range wireless mesh network with 1 exit node connecting to a desktop PC

to incoming user requests will allow us to easily allocate users services in a manner that alleviates strain on one particular node in the network. This also enables the subject of investigation of distributed storage through the pooling of storage resources on the network.

But the very nature of this network puts very severe constrains on the type of technologies that can be deployed. Should the functionality provided by the network infrastructure need to be expanded beyond provisioning of a transport layer this becomes an issue. Low power devices typically have two characterising traits, i.e. low processing power and low memory capabilities. Which means whatever solutions is developed must when, when deployed, operate with care when allocating services.

This very nature though becomes beneficial when the context of their use changes. With no moving parts, the risk of equipment breaking down decreases. The low power constraint now also turns into an advantage when one looks at deploying devices where constant power supply is not an option. Making these the perfect devices for unattended installations.

With the distributed storage system and service provisioning layer in place, users will be able to connect to the network and interact with the services. Users with

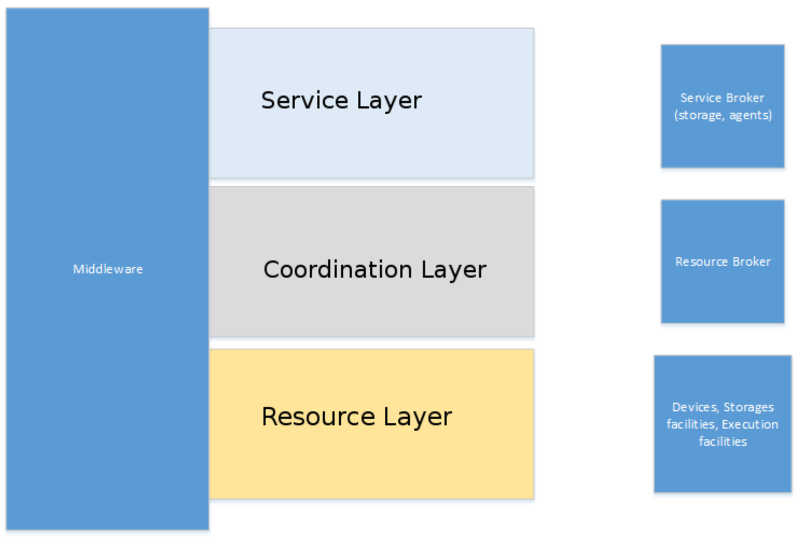

Middleware
Service Layer
Coordination Layer
Resource Layer
Service Broker (storage, agents)
Resource Broker
Devices, Storages facilities, Execution facilities

Figure 2.2: The envisioned architecture layers within the middleware stack

smart phones or laptops will be able to wirelessly connect to gateway devices and access an expanded remote storage. Or perhaps even connect to one of the various services running on the network.

2.3 Network architecture

It is understood, as described in the background reading described in the previous chapter, that a middleware designed to manage our community clouds or cloudlets would need to be split up into three discrete areas of operation. Figure 2.2 presents a broad overview of this envisioned architecture.

Instead of the resource layer being the intermediate layer in-between the coordination and service layer it is instead put here as the primary layer. We shift the predefined meaning of resources as implied by the background to instead relate more towards raw resources and coordination to imply to the direct management of these resources and users in the network. Here our resources consist of the individual nodes and the services that they will provision to the network. This can consist of providing storage facilities, data warehousing where database capabilities are made available or locations where certain services can be executed.

The coordination layer consists primarily of a resource broker. The resource broker is responsible for keeping tabs on the current performance status of all nodes in the network. Upon the resource broker receiving a request for services, it can then determine if there is a host in the network that is capable of handling the requested task, and then subsequently allocate the task to the host if one is available.

The service layer acts almost as a meta layer, since the services are already provided in part by the nodes themselves. The resource broker acts as a gateway between the end user/service and the desired resource it wishes to access on the network.

The project itself in this sense sees two distinct areas in which service provisioning occurs. The allocation of access to services such as webservices and software solutions running on nodes. The other is the storage capabilities that are enabled by aggregating storage space from each node in the network.

It was determined that only one of these areas should be tackled for a work of this nature and so service provisioning will primarily be the subject of 3 chapters of this work. The subject of storage will be expanded on further in this chapter, but will primarily remain the subject of a future work.

The allocation of services is based on the resources available on the various nodes within the network. This means that resource monitoring must occur before any allocation of a service can be conducted.

2.3.1 Agents

At each layer of the network we can think of various agents working in concert with each other. Here agents are defined as independent services running on each node in order to provide overall goal. In our case our middleware aims to provide a means of managing our networks to ensure optimum performance.

Here an agent is defined as a process that runs in a network highly coupled to their environment, in a sort of autonomous fashion. Here they will perform various tasks on the network, instead of network administrators to achieve the task of maintaining a healthy networked environment.

We shall consider the network to consist of the following agents:

1. Cloud Coordinator:

   - Co-ordinates all the resources in our cloud.

- Responsible for allocation of services and resources for services.

2. Storage Coordinator:

   - Resides at the cloud coordinator.

   - Allocates storage and services storage requests.

   - Aggregate storage resources in the network.

3. Service Coordinator:

   - Resides at the cloud coordinator.

   - Aggregates and allocates services to the various nodes in the network.

4. Hash Agent:

   - Resides at the cloud coordinator.

   - Allocates unique identities to nodes in the network

5. Knapsack Agent:

   - Resides at the cloud coordinator.

   - Responsible for calculating optimal nodes in the network for servicing various requests.

   - Calculates values based on values received from client nodes in the network.

6. Monitoring Agents:

   - Resides on all the clients (the cloud coordinator is also a client of itself).

   - Responsible for sending system values from the client to the cloud coordinator. - Returns values only if the node notices a trend which indicates a difference in performance values.

7. Discovery Agent:

   - Resides on all nodes in the network.

   - Discovers neighbours on the network.

   - Responsible for the detection of nodes on the network and selecting the node that will be in charge of coordinating the network.

   - Responsible for selecting a new cloud coordinator when the current coordinator goes down.

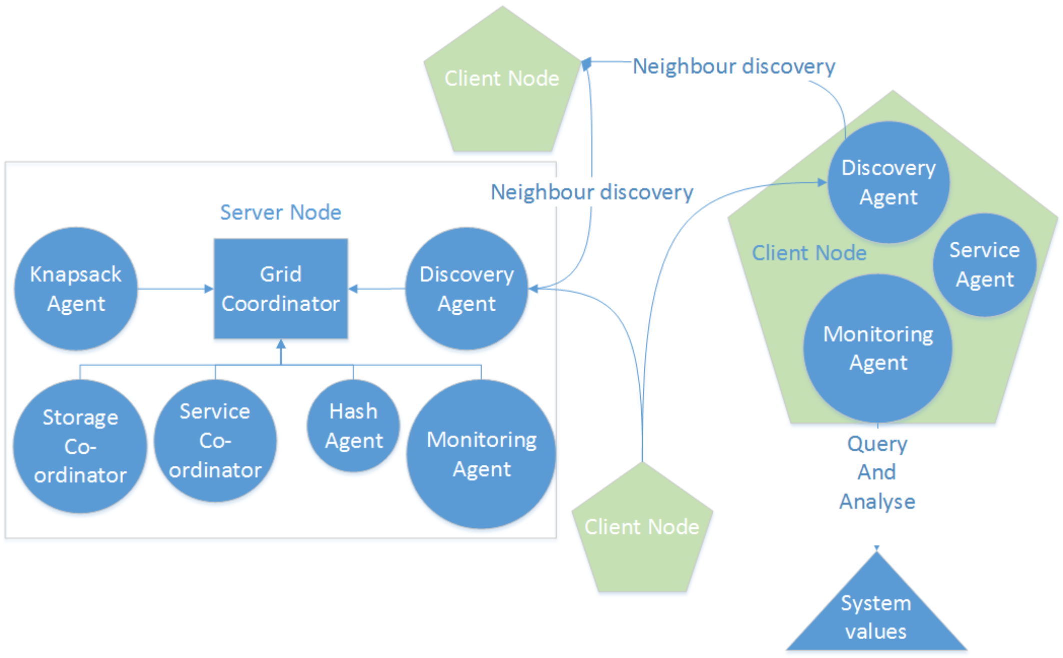


Figure 2.3: Agents in concert in a participatory cloud network

8. Service Agent:

   - Resides on all clients.

   - Responsible for the injection of various service modules into the service list available on the network.

Figure 2.3 demonstrates all these services in operation in a network. While being decoupled from each other, they still require the presence of other agents in the network to be effective in anyway. As stated in the service list, the coordinator of the network acts as an agent of the network itself. So it too can take part in the provisioning of services, if deemed appropriate by the administrator.

All these services working in concert with each other demonstrate what will be referred to here as the Community Cloud Management Protocol, CCMP. This is further highlighted in Figure 2.4. We can see it being composed of two distinct sub-protocols; i.e the Lightweight Network Monitoring Protocol, LNMP; and the Lightweight Resource Allocation Protocol, LRAP.

The investigation into these two distinct parts is examine further in chapters 3 and 5. What we have not mentioned yet, is the database layer which is somewhat

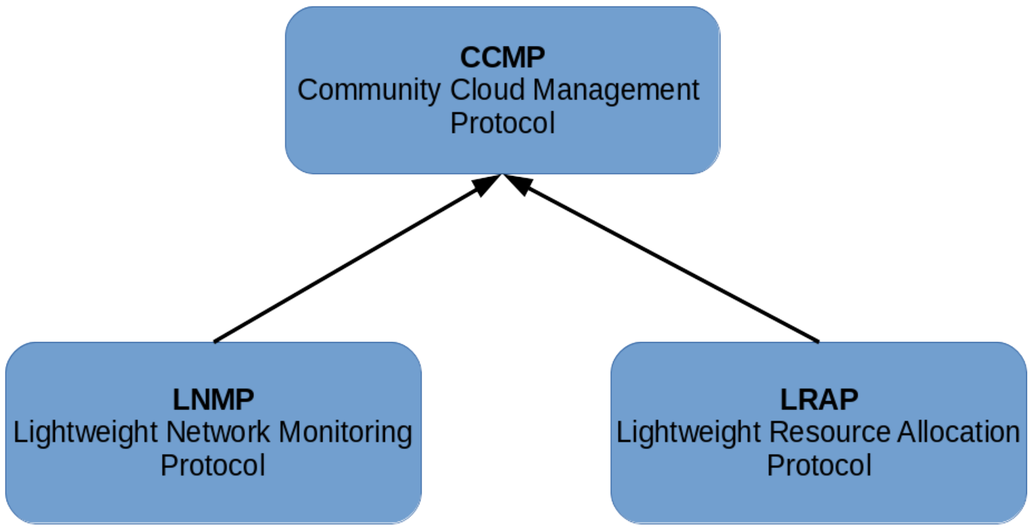

CCMP
Community Cloud Management Protocol
LNMP
Lightweight Network Monitoring Protocol
LRAP
Lightweight Resource Allocation Protocol

Figure 2.4: The community cloud management protocol

abstracted from this diagram which manages the data storage in the network. This is explained in more detail in chapter 4.

Many of the agents mentioned above overlap in the two protocols and they are essentially part of one but split up because the functionality can be modularised. LNMP, can be considered the collection of agents forming part of the overall network monitoring framework built in order to facilitate client monitoring, while LRAP can be seen as the collection of server side agents that are required by a variety of services or processes to perform allocation operations

2.4 System Monitoring

The correct system metrics must be collected in order to best indicate the performance status of the particular nodes in question. The CPU and memory usage was determined to be the best metrics with which to perform performance evaluation.

Memory includes both primary and secondary memory, that is RAM, SWAP; and hard drive storage (this information becomes necessary when storage services are accessed). CPU usage consists of various metrics depending on what operating system one uses.

Figure 2.5 demonstrates 2 different means of calculating CPU usage. These

$$available\_cpu_{load\_avg} = (\frac{1.0}{Unix\_load\_avg + 1.0}) * 100\%$$

$$available\_cpu_{vmstat} = (idle + \frac{user}{rp + 1} + W * \frac{system}{rp + 1})$$

Figure 2.5: 2 methods for calculating processor load

formula are extracted from Rich Wolski et al.'s work regarding the predicting of CPU availability of Time-shared Unix systems on the computation grid[21]. In the formula, 'rp' is the run process length, or rather the number of processes running in on the system.

The value for load could be used, but this value is only available for Linux machines. Even if the solution was to be deployed in a Linux only environment, load averages do not give nearly enough information about the CPU state. Machines are known to report very high load averages simply because of processes that have broken or are waiting on IO operations but are still in the CPU run queue (This adds to the load while not imposing any additional performance penalty to the machine). The values are also not normalised for the number of CPUs on the system, so knowledge of that also has to be obtained before hand. The first example is discarded as the load average does not provide a good enough metric to determine CPU activity.

It is better however, to select a combination of the CPU idle time, along with its user(usr) and system(sys) times and perform calculations based on this. This value is more universal for Windows and Linux operating systems.

The second formula however, does not sufficiently penalize CPU activity on the hosts enough. The primary function of the nodes in the network are to act as gateway devices and provide a transport layer for the wireless sensor data being collected. Since this is the case we have to ensure that when executing tasks on a particular node, that it has enough resources available to perform its primary functions in the network. As this is the case, the formula is modified and illustrated in figure 2.6.

The system also stores the values for the available swap area, if available, as well as the amount of free RAM on the host machine. These values can either be stored in a database or held in memory for a limited amount of time. The current solution

$$available\_cpu_{vmstat} = (idle - \frac{user}{rp+1} - W * \frac{system}{rp+1})$$

Figure 2.6: Updated load processing formula

is set to hold values for up to 60 minutes or 60 measurements (one measurement taken every 60 seconds); where after 60 minutes the first value inserted is the first value removed from the list.

Various database options exist. If this is a local only database or no database, updates should be sent to the cloud coordinator at regular intervals (30 minutes) or whenever an event occurs where it is determined that the load has spiked beyond a defined value. This value is selected by calculating the standard deviation of system values for the last 60 minutes. If the newly monitored value falls within the standard deviation, the updated value is not sent to the server; otherwise it is.

This means that the server will most likely receive quite a few updates from the node when the node is brought online for the first time as it should still be in the process of trying to find its median operating performance values. These calculations are carried out individually for RAM, CPU and SWAP values. If any of these values indicate a spike an alarm is triggered and sent off to the server.

The SIGAR library is used here to collect the system resources values. The justification for this decision is expanded upon further in chapter 3. Numerous tests are conducted which indicate that LNMP, using SIGAR, proves to be a more lightweight solution in its application than SNMP for network monitoring.

2.5 Database

The selection of BigCouch as the recommended distributed database solution was a complex one. We needed a method that put as few memory constraints on the system as possible, as the devices used were limited in their memory capabilities. The database also could not be wholly contained within a single node.

Initially a solution considered where SQLite was installed on each host and each host contained its own database. The manner in which this was to be made into a distributed database proved to be too complex and was beyond the research questions required of this task

Using BigCouch though means that no database drivers were available at the moment of writing of this thesis as the solution is fairly new. An attempt was made to use the CouchDB drivers but when noticing that the drivers did not support view operations, this was abandoned. In BigCouch\CouchDB a view is a JSON object with functions as properties of that object which allow you to access your data based on filters you can specify in these functions.

Since BigCouch allows any sort of document to be stored, users need to ensure they are getting the correct documents. You can specify a document type on storage of your unique data and retrieve your data from the database based on the document types specified. Note though, that this is not some unique property inherent to the document. All data is stored in JSON and defining a document type is as simple as setting an attribute on a JSON object.

Database connectors were developed using *curl* since BigCouch, like CouchDB, expose a Web API which allows a full range of interaction with the database by sending various HTTP messages along with the content to alter the database state. *curl* is a command line tool which allows transferring data with URL syntax[22].

The database agent though provides an abstraction layer to all connecting systems which require database access. So technically this could easily be replaced with connectors to any other sort of database. Configuration file as references in Appendix A, can be configured to select any database host to store the data. Selecting of a different database technology can be selected by configuring the 'database_type_string' field. Currently changing this in configuration file has no effect, but future support for it has been enabled.

2.6 Service Allocation

Service allocation primarily falls within the operating functions of LRAP. That is the allocating of services to incoming requests. The applicability of this solution is tested out in chapter 5 with very promising results being presented.

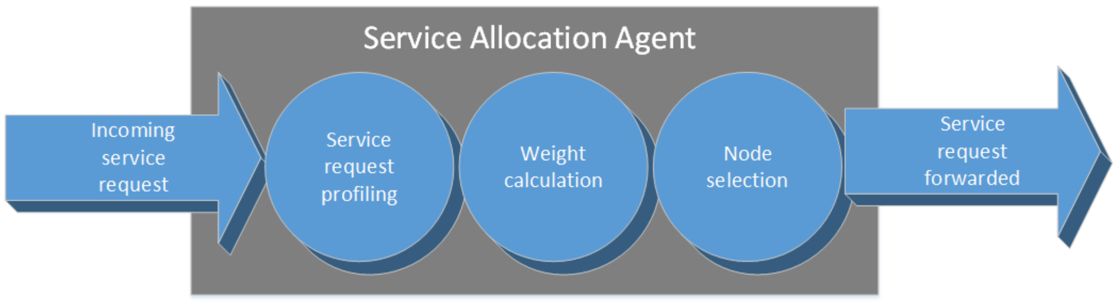


Figure 2.7: Service request flow

Allocation would come as a direct response to services being requested in the networked environment. It should be noted that this is the behaviour being explored instead of a pre-emptive service management infrastructure; where processes that are already running on certain nodes are migrated across the network when the appropriate changes arise. On a pre-emptive infrastructure, a task will be moved from node A to node B if the usage on node B runs a certain percentage or level lower than node A. This sort of infrastructure makes more sense when one is performing raw computing tasks such as that in the SETI project.

Incoming service requests would access a networked or locally global API which would either return the location of the host where the request could be executed or forward the request to the particular node/service that the user application has requested. This is of course only in the circumstances that the requests made are actually successful.

The allocation process developed in this project took inspiration from the greedy approximation algorithm which is used to solve the knapsack problem. The knapsack problem is described as being given a number of items, each having its own weight and value. The idea is that the knapsack is of finite size and the desire is to have the most optimized ratio of value and weight in the bag.

In our case each host on the network has a finite amount of resources available. If one examines the configuration file in appendix A you can see that a 'penalty_max' value has been defined. This determines the maximum load allowed on the node in question, i.e. the finite size of the bag.

Not only that, the "bag" is not empty to begin with. These nodes are running their operating systems with whatever processes they need to be active, let alone the LNMP daemon which is collecting system performance statistics as well as the database layer (This performance impact is negligible when looking at the heap values in chapter 3).

Each service running on the network typically consumes a certain amount of resources, such as CPU and memory. For example, loading a web page via Apache2 would place certain constraints on the disk, memory and CPU. These can all be measured and with it an average can be calculated to determine a resource consumption profile for the application in use.

This profile can be translated into a weight that the process would be typically associated. This weight though should typically consist of multiple variables as each process could have different sort of impacts on the system. For example, some process might consume more RAM whereas another would be more CPU intensive and some might even consume a lot of everything available on system.

Working with this sort of multidimensionality was a bit too complex for this work and was not the focus, so instead a broader single valued weight was decided upon for each process in the system. In our examples used we simply applied integer values which had no upper bound, but applying. The reader is now also directed to appendix B where a sample service definition file has been stored. Each service has a name, weight, port and URL associated with it.

It shall be noted however, that in order to better allocate services to nodes, the multidimensionality of the service resource requirements and that of the systems involved has to be brought to the table as this will only make the allocation algorithm all the more robust. This too is marked for future work.

Each node on the network is thus offering its own bag or 'execution bin' where services can run. Instead of trying to fully optimize one bin, the filling of the bins is distributed by filling the least empty bin at the time of each insertion. This outlined in the following steps.

- All the hosts on the network are connected to the cloud coordinator. The coordinator ensures that for all the nodes connected it has the latests stats concerning their performance status.

- It then arranges these hosts in ascending order by how much absolute availability they have. This means that if a host is connected and it has a much higher 'penalty_max' value defined than another node on the network, the chances are much more likely that the node with the higher maximum penalty defined will be selected to be on top of the list.

- The node connects with all its available services to the cloud coordinator and shares this list. The services are then added to the connection pool and the list of hosts associated with particular services is updated. The nodes that connect have their penalty values adjusted. This is done by the following method.

- A value out of 3 is calculated for the system performance. The proportion of non free to total RAM is calculated. This can go to a maximum of 1. The same procedure is carried out with SWAP space as well as CPU idle time as well. If no swap area is defined then the system defaults to selected 1 as the swap value. This was deemed appropriate seeing as the lack of swap area leads to the system being more resource constrained.

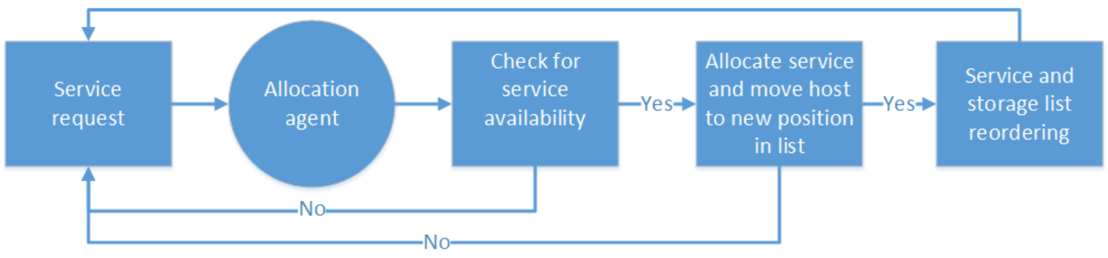

Service request
Allocation agent
Check for service availability
Yes
Allocate service and move host to new position in list
Yes
Service and storage list reordering
No
No

Figure 2.8: Allocation cycle

- These values are added up and divided by 3 to get a percentage of usage for the system. Here, in this case, 3 represents a system that is completely loaded; 0 indicates the inverse. This same percentage value is then used to get the percentage of the maximum penalty which is used as the user's current penalty value.

- When services are requested from the allocation server, it proceeds from the top of the list to determine firstly if the node has the particular service being requested and then if the first node that it has encountered has the available

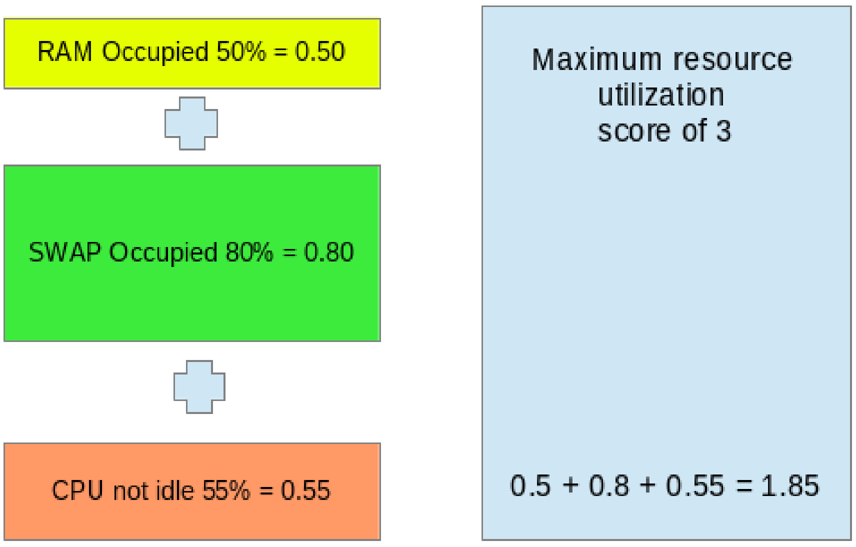


Figure 2.9: Sample allocation resource calculation

resources for the service to be executed on that particular node. This is done by adding the weight of the requested service to the current penalty value of the queried node. If the new weight exceeds the maximum penalty then the service request is rejected. Should it be successful an address is returned.

- Thereafter the currently penalty is readjusted by adding the weight until the next evaluation cycle (every 60 seconds) completes and the new penalties are adjusted based on any new performance values retrieved from the client nodes.

Testing of the solution proved itself to be a somewhat complicated procedure as it becomes pointless to test on its own without complex simulation so a real life implementations was decided upon. A DNS server was considered the perfect application to test the system. This also become a crucial part of the infrastructure, at least for the current vision of the system.

It was determined that each service on the network would be requested by its unique domain. We did this by defining our own network domain for services, '.site'. Services would be referenced by adding the service name as the subdomain on the network. That is, to accesss the Asterisk service on the network one would query it with 'asterisk.<network_domain>'. This would allow easy setup of the DNS server to capture all service requests made in the network.

The DNS server would in tun then interpret the messages received from the domain request and determine if a particular request was valid by first checking the network domain. Should this value not fall within the designated domain for the network then the request will be immediately rejected. So if we define our search domain[see appendix A] as '.site' and a request comes to the DNS server for 'asterisk.local', the DNS query will be rejected. Should this process occur with a valid domain name, the query is then forwarded on the the LRAP agent with the extracted service name, i.e. 'asterisk'. The LRAP agent would then return the UUID of the host that would be able to fulfill the request.

The DNS server has to then query the network agent which manages all the connections and tracks information such as UUID to IP address mapping. This will then return an IP address which is in turn retuned from the DNS server.

One more dimension which has not been alluded or inferred upon so far in this work, is the particular network topology involved with these calculations. All the service operations involved in the network require the transmission of data via the network. This data transmission applies its own load on the various systems involved in the network and when topology is not considered, nodes could be selected would require datagrams to pass through nodes that are already burdened with their own tasks.

This could lead to cases where the nodes being traversed become overly burdened, or situations can occur where nodes packets are dropped or delayed due to the increased load being put on the network. In a performance sensitive context such as VoIP, this becomes a very crucial determination step. This performance graph of the network will also remain the subject of future investigation.

2.7 Data Storage

This section was not included in the draft implementation of the architecture developed. This was deemed infeasible due to time and resource constraints. It is included as time spent developing the solution and may serve as guide for possible future work.

Storing data in the cloud will be a slightly different procedure to service allocation as we wish to ensure that data when stored is done in a way that is both secure and redundant. This requires two concepts, replication and security. Security on lightweight devices is not to be taken lightly as they are small and the processing power is very limited.

We will limit ourselves in this context by applying our techniques in a very limited manner, by only considering Alix Boards, to take full advantage of the on-board cryptographic accelerator on these devices. The Alix board crypto accelerator only works with AES 256 CBC. In an ideal situation the encryption standard would be negotiated between the devices (perhaps a device performance rating could be incorporated in the communication and the device with the lower performance/higher priority can get preference over the encryption techniques used.)

Redundancy measures will need to be implemented to ensure that if one particular node goes down or leaves the network that the files stored thereon should be fully recoverable from other nodes on the network. Figure 2.10 illustrates the storage node selection procedure.

Each host that joins the network will have in its configuration a predefined *n-copies* variable. This value determines the number of copies to make on the network for each file storage request.

Files will be broken into 100kb blocks, encrypted and distributed through the network to a number of hosts. The top three available hosts in the network are selected as the storage repository. This is recalculated for every storage write request.

Even though a host might have available storage space, the file might require additional hosts just to store one copy of a file. The idea in this scenario was to add up the total storage available in the network and divide that by the size of the file. Should the answer be less that n-copies (the required number of times a file needs to be duplicated in the network) then the request to store the file should be rejected.

There is also the notion of including a maximum file size based on the available storage in a network. This means that as files are being added to the network there is a downward sliding scale which will negatively impact the file storage sizes in the network.

There is also the impulse that this file size should be determined by the nth-copy node. I.e if we select that we wish to have our redundancy set at 7 nodes, then the 7th largest node in the network (in terms of storage space) will be used as the bar for selecting the largest file size limit.

The request to store the file should be rejected if the number of copies stored in the network is less than the defined *n-copies* value.

Initially the thought was to provide a storage facility for computational and service activities that would be carried out in the grid. A cloud file system implemented on a network such as this will allow for a much richer experience in terms of the applications that can be developed and will also aid in the increase in accessibility of data in a large distributed network of nodes.

While it might be possible through the use of smart streaming techniques, working with files that are much larger than the available system memory would most likely be an incredibly difficult task. This left out of the feature specification for this part of the project.

Initially a ratio of 3(nodes) to 1(file) was envisioned as a sufficient enough to ensure availability in the network. Making this procedure dynamic or customizable per node can be looked at at a later stage. For the sake of simplicity, the *n-copies* variable will be set to a hard limit of 3 in the configuration files on each host.

2.8 Security

It must be equally duly noted that no security means have been included in this first draft of the project. It has not been possible to expand on these features but it must be emphasised the requirement for it to be incorporated in order for this service to become useful in real networked environments.

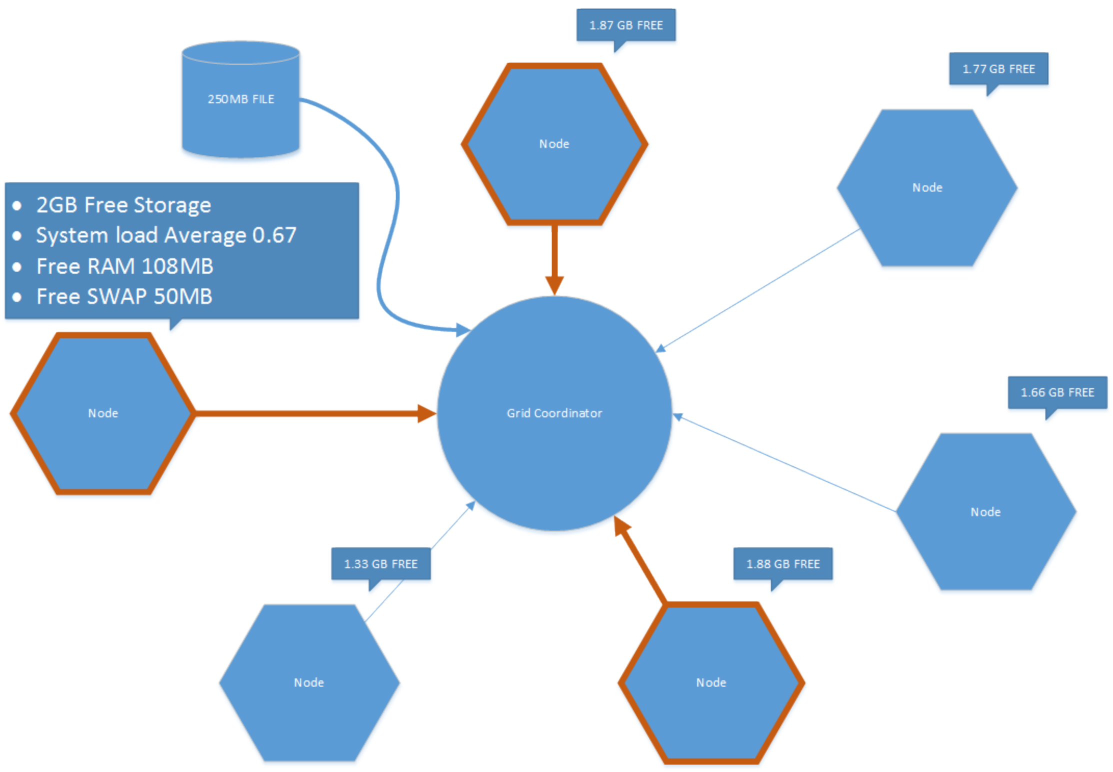


Figure 2.10: Storage selection procedure

Chapter 3

Lightweight Network Management Protocol

The simple network monitoring protocol is currently seen as the de-facto standard for network monitoring in IT settings. It is usually used in conjunction with tools such as Cacti, Zabbix and Zenoss to provide most of the features that IT professionals require to efficiently monitor network systems. However, SNMP is not be suitable for the emerging ubiquitous networking application environment where energy constrained devices operate with limited storage and processing capabilities. These devices are usually deployed unattended and sometimes with intermittent power supply. Furthermore, the client server model underlying SNMP may create traffic bottlenecks and a single point of failure for the underlying monitoring systems. Building upon these limitations, we aim to showcase LNMP and demonstrates its relative efficiency when compared to SNMP. Preliminary experimental results reveal that LNMP simplifies the network monitoring process and provides a lightweight approach in monitoring low power network devices.

Abbreviations

SNMP: Simple Network Monitoring Protocol (Version 3)

LNMP: Lightweight Network Monitoring Protocol

SOHO: Small Office, Home Office

IoT: Internet of Things

MIB: Management Information Base

ASN.1: Abstract Syntax Notation 1

3.1 Introduction

Network monitoring is a critical part of large corporate IT infrastructure, although it extends far beyond that. Almost every network that extends its reach further than the typical home or even Small Office/Home Office (SOHO) environments require some sort of network monitoring system. This is due to the fact that critical IT decisions need to be taken based on network performance metrics. In this sense, the term network monitoring does not only relate to network throughput but also to the health and performance of the actual equipment on the network.

With the growth of complex infrastructure, Internet-of-Things (IoT) and the multitude of smart devices that connect to networks to access various services, the need for reliable network monitoring is even more pressing.

3.1.1 The Simple Network Monitoring Protocol (SNMP).

SNMP [23] [24] [25] was created in 1988 and has largely been used as the standard tool for network monitoring. SNMP is built upon a client-server architecture where i) physical resources are represented by managed objects and ii) collections of managed resources are grouped into tree-structured management information bases following the ASN.1 format.

The user installs the SNMP daemon on a particular machine which is then configured to have some local or remote process to connect to it. It is then able to reply to requests for data, which usually come in the form of community strings and user credentials. Usually in large scale IT deployments this is done at one central hub and all devices are accessed remotely. This process is not limited to data collection. SNMP is able to modify, albeit in a very limited manner, the behaviour of processes on the devices by sending a SET request to the particular device. This collected data is then usually converted into readable graphs that can be interpreted by the IT staff managing the network.

In very large IT deployments, scenarios can arise where the central SNMP monitoring hub can become overburdened with traffic data from hosts due to the sheer number of devices on the network and the large volumes of requests and data that would need to be collected. Furthermore, a failure of the central hub may become fatal to the whole IT infrastructure that might become unavailable until repair, maintenance and/or replacement actions are undertaken. Moving forward we need to re-examine what software solutions we will have to deploy in order for us to make the most efficient choices for monitoring our networks while ensuring that our attempts at mitigating problems do not cause problems itself.

Mobile agents [23] have been suggested as a method to mitigate the circumstances outlined above. In the process outlined in [23], SIGAR is used instead of SNMP to monitor system performance. Here the values are collected on the nodes by an agent running on the local machine and when the system or network administrator wishes to find out more detail about the machine(s) in question the data is then transmitted from the target machine to a server which in turn displays that collected data to the user.

3.1.2 Lightweight Network Monitoring Protocol (LNMP)

The system information gatherer (SIGAR) [26] is a cross-platform API for collecting software inventory data. It provides auto-discovery functionality and can be extended to add new functionalities to a network management which could not be possible with simple network management protocol (SNMP). SIGAR is a platform independent API that provides support for Linux, FreeBSD, Windows, Solaris, AIX, HP-UX and Mac OSX across a variety of versions and architectures.

Its main offers to users and developers includes portable access to inventory and monitoring data. These include i) System memory, swap, CPU, load average, uptime, logins, ii) Per-process memory, CPU, credential info, state, arguments, environment, open files, iii) File system detection and metrics, iv) Network interface detection, configuration information and metrics and v) Network route and connection tables.

The lightweight network monitoring protocol (LNMP)[18] is a monitoring tool which is built around the functionality of SIGAR to provide a means of network monitoring of cloud infrastructures underlying community networks. Its lightweight processing capability and agent-based architecture enable more flexibility compared to the client-server SNMP architecture in terms of deployment, storage of the data that is gathered in the network.

The clients themselves on the network thus become responsible for monitoring themselves and detecting noticeable changes in the operating parameters. Only noticeable changes are reported, this in turn reduces load on the network when compared to SNMP by not having the hosts continuously reporting to a single point on the network. In the proposed network management architecture in figure 2.4, we seek to describe a system that is able to effectively monitor and respond to system demands. The system aims to be lightweight in its design in terms of its CPU and memory consumption but also in the way it consumes very little resources on the network. The LNMP software architecture depicted by Figure 1 includes various agents described below.

3.2 Experiment Evaluation

We conducted a number of experiments to compare SNMP to the newly proposed LNMP in terms of network monitoring under both Linux and Windows operating systems.

3.2.1 Experimental Setting

It should be noted the extreme difficulty of gathering system data with SNMP compared to the LNMP. To create code that is immediately cross platform portable, one needs to ensure that all the MIBs (Management Information Base) are in place. Then implementing accurately the querying of the correct object identifiers (OIDs) for each operating system as these OIDs differ for Linux and windows platforms. This is due to each operating system having different interpretations of system resources.

For this experiment we conducted a simple comparison between values retrieved from the CPU (user and kernel utilisation) and RAM usage values on Windows and on Linux systems. This immediately proved to be not straightforward at all. The only values that could be retrieved on windows relating to the CPU was the CPU load.

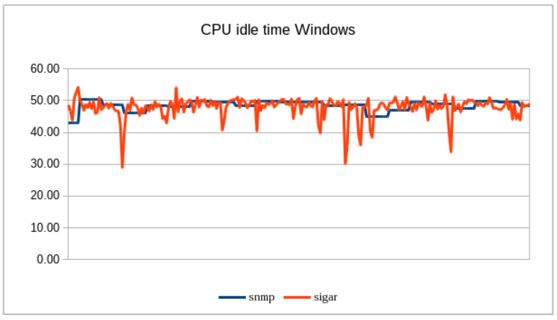
This is differs from load value collected from Linux as this is only a percentage statistic based on the amount of time that the CPU was not idle in the last minute. The value presented was also only for only one particular core and thus separate queries have to be made for each core in order to calculate the total usage value for a multi-core processor. When one compared this to the values in the task manager, it was shown that the values did not match up. This was due to SNMP gathering the average values of the system over the course of 60 seconds.

We contrast this with the collection process on Linux. Upon querying idle CPU time, only one value is returned. The returned value on linux is a 60 second value that is calculated from the moment the query is made.

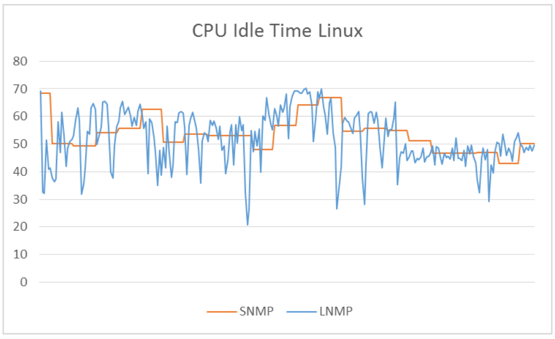
CPU idle times.

For the first experiment we decided to compare CPU idle times retrieved by the LNMP and SNMP libraries. This was the simplest experiment to conduct as the hrProcessorLoad (1.3.6.1.2.1.25.3.3.1.2) OID gave us the average, over the last minute, of the percentage of time that this processor was not idle. According to [25], the return value might be subject to a one minute smoothing if necessary. This value is then subtracted from 100 to get the percentage of the idle time. This value has to be calculated for all cores on the particular system so knowledge of the system specifications are required before hand.

With LNMP we simply had to execute the query the SIGAR API for CPU idle which returns the total time (all cores) the system spent in the idle process.

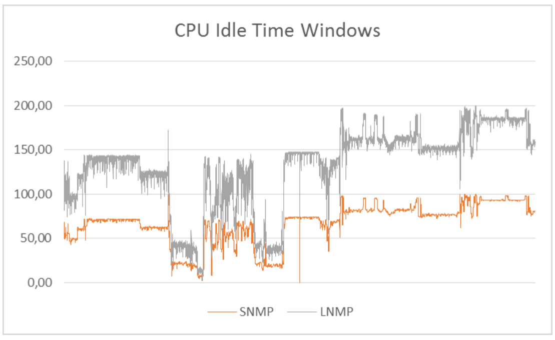


(a) 20 minute monitoring window, Windows

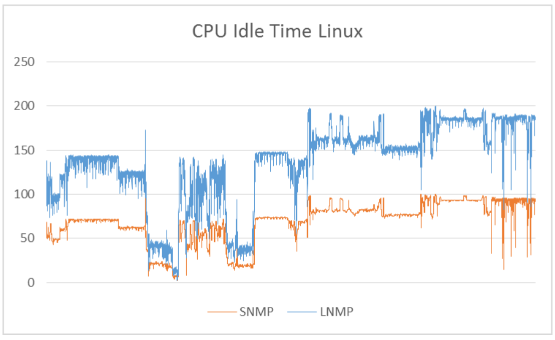


(b) 20 minute monitoring window, Linux

Figure 3.1: Idle CPU times for Windows and Linux



(a) 24 hour monitoring window, Windows



(b) 24 hour monitoring window Linux

Figure 3.2: Idle CPU times for Windows and Linux over a longer time period

Total RAM values.

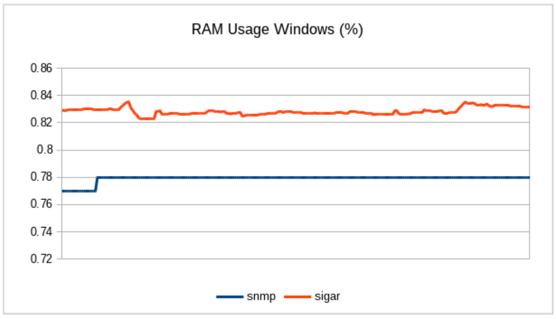
During our second experiment we discovered that, when trying to query the value of total RAM used for Linux hosts with SNMP, the discrepancies were noted. We compared the results returned when calls were made using the GNU 'free' utility and the results returned when querying SNMP for returning the value for the amount of RAM being used. (This value was queried because SIGAR only provides calls for the total amount of RAM available and the amount of RAM being used). The results returned from LNMP and 'free' matched up but not with the SNMP values. This can only be reduced to possible error in the SNMP libraries as all other queries returned the correct values. The work around was to instead query the amount of free RAM via SNMP and subtract that from the total RAM value.

The SIGAR API reports only current system readings contrasted to SNMP which collects averages over 1 minute. This same behaviour is emulated with LNMP where values are collected at 5 second intervals and then averaged out. This was to get rid of any variance spikes that could be picked up.

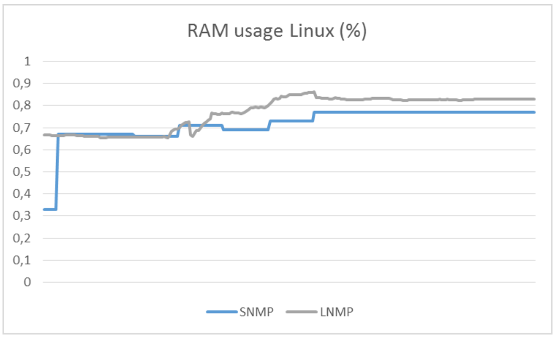
The values retrieved from SNMP were rounded off to two decimal places. The same effort was made with the LNMP protocol as SIGAR by default gave values that had 10 decimal places. Both test environments were Ubuntu and Windows XP virtual machines (VirtualBox) running on the same host machine at the same time. Specifications of the host machine are: HP Pavilion dv6-3124si, AMD AMD Phenom II Quad- Core Processor N950, 6 GB DDR3 RAM, 500 GB Seagate SATA 2 HDD.

3.2.2 Computation Accuracy

Figures 3.1a and 3.2a illustrate the collected statistics on a Windows operating system over two different time periods. Figure 3.1a illustrates the variance that in the readings when collection of performance values through LNMP occurs. The peaks and troughs though, show that the data is following a similar pattern to what the SNMP library is reporting. This is confirmed when looking at Figure 3.2a which simply two stacked graphs for the reported LNMP statistics to better visualise the

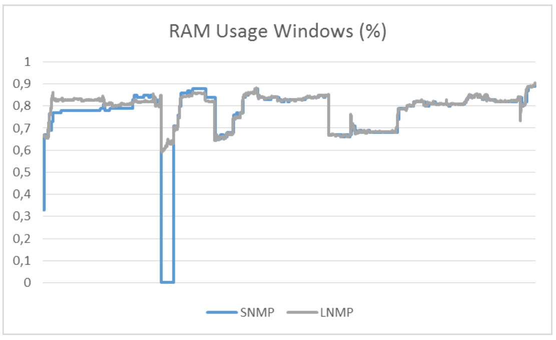


(a) 20 minute monitoring window, Windows

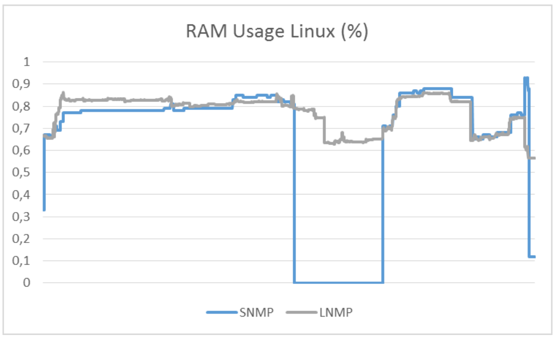


(b) 20 minute monitoring window, Linux

Figure 3.3: RAM readings for Windows and Linux

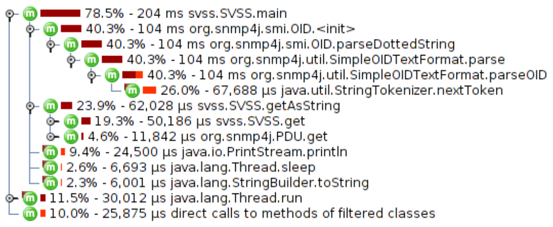


(a) 24 hour monitoring window, Windows



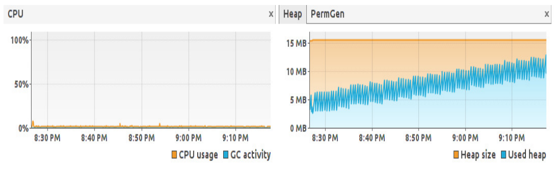
(b) 24 hour monitoring window Linux

Figure 3.4: RAM readings for Windows and Linux over extended time frame



```
78.5% - 204 ms svss.SVSS.main
  40.3% - 104 ms org.snmp4j.smi.OID.<init>
    40.3% - 104 ms org.snmp4j.smi.OID.parseDottedString
      40.3% - 104 ms org.snmp4j.util.SimpleOIDTextFormat.parse
        40.3% - 104 ms org.snmp4j.util.SimpleOIDTextFormat.parseOID
          26.0% - 67,688 µs java.util.StringTokenizer.nextToken
  23.9% - 62,028 µs svss.SVSS.getAsString
    19.3% - 50,186 µs svss.SVSS.get
    4.6% - 11,842 µs org.snmp4j.PDU.get
  9.4% - 24,500 µs java.io.PrintStream.println
  2.6% - 6,693 µs java.lang.Thread.sleep
  2.3% - 6,001 µs java.lang.StringBuilder.toString
11.5% - 30,012 µs java.lang.Thread.run
10.0% - 25,875 µs direct calls to methods of filtered classes
```

(a) CPU profile (runnable) of SNMP monitoring.



(b) SNMP CPU and Heap values.

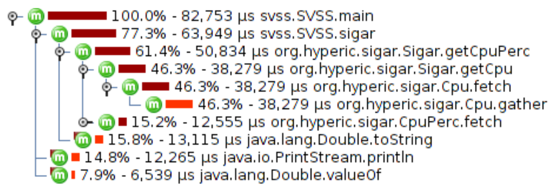
Figure 3.5: Profiles of system running SNMP tests

rise and fall of system readings across the two monitoring tools. Figure 3.2a reveals that the reported values are identical in the trends in idle CPU time.

The LNMP readings appear much thicker due to the erratic nature of the values being captured, but average around similar values to those reported by SNMP. This can be seen in figure 3.1b and 3.2b which display the Linux monitoring process, the results yielded similar results to those achieved in the Windows environment. They also reveal that on immediate polls of gathering system data, the LNMP suffers from erratic spikes once again when examined over a shorter time-frame in more detail.

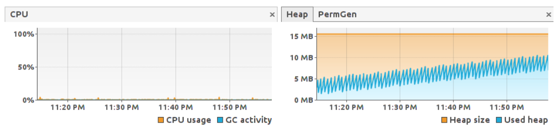
We have also included the results we received while experimenting with the exact same two monitoring protocols but when monitoring the RAM usage. As noted above, there was a lot of discrepancy in the results we received back from SNMP. The values would fluctuate quite wildly at times from what the system was actually reporting. At times the values would drop straight down to zero (i.e. no RAM being used) during normal operation, which should not be possible at all.

The values at times would also not match up with the values being reported by the system utilities. We noted that the LNMP values offered the most constant values when compared to the retrieved system values (this is probably due to LNMP querying the system directly). This issue with SNMP was apparent on both Linux and Windows test environments as can be seen in Figures 3.4a and 3.4b. It also appears that SNMP also implements a 60 second smoothing window on the RAM data as it does with the CPU data. The system was regularly queried via use of the conventional system administration tools to verify correct values were being collected.



```
100.0% - 82,753 µs svss.SVSS.main
  77.3% - 63,949 µs svss.SVSS.sigar
    61.4% - 50,834 µs org.hyperic.sigar.Sigar.getCpuPerc
      46.3% - 38,279 µs org.hyperic.sigar.Sigar.getCpu
        46.3% - 38,279 µs org.hyperic.sigar.Cpu.fetch
          46.3% - 38,279 µs org.hyperic.sigar.Cpu.gather
      15.2% - 12,555 µs org.hyperic.sigar.CpuPerc.fetch
    15.8% - 13,115 µs java.lang.Double.toString
  14.8% - 12,265 µs java.io.PrintStream.println
  7.9% - 6,539 µs java.lang.Double.valueOf
```

(a) CPU profile (runnable) of LNMP monitoring.



(b) LNMP CPU and Heap values.

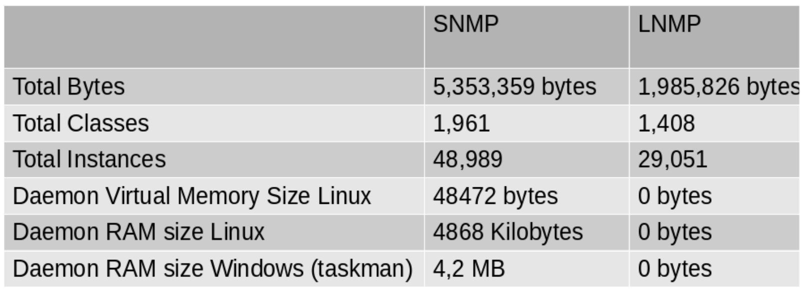
Figure 3.6: Profiles of system running LNMP tests

3.2.3 Computation Time

Calculating the computation and resource requirements of the two protocols required us to perform some profiling of the software as well as examining the system resources required for running any additional system resources needed. For this purpose, we used the following tools: Windows task manager, htop, JProfiler 8 and VisualVM. VisualVM (bundled with JAVA 6 and up) and JProfiler (proprietary) are profiling tools which allow us to examine the memory and computation costs related to the software solutions used in the monitoring process.

The solutions require the system to be calibrated first which involves running the profiling software when then system is operating at a stable equilibrium. It then stores this data as the mean operating values (i.e. CPU and memory size) for the system and any overhead caused by the profiled software solution is then gathered by subtracting the mean operating values from the current operating values. We then ran our software solution twice, first we got the LNMP performance values from the system. Then this process was terminated and subsequently followed up with monitoring the system with the SNMP daemon running in the background and the solution with the SNMP library now being used to monitor the system statistics.

We proceeded to change the software solution so that it would attempt to query the system for values every 500 milliseconds. This way we could get a better view on how the system is actually performing when making these calls to the system. It must be noted in the results that the times displayed are for the runnable portions of the threads. Times were slightly longer when considering the waiting time by the threads. SNMP queries went via the network, even when on the local machine,



	SNMP	LNMP
Total Bytes	5,353,359 bytes	1,985,826 bytes
Total Classes	1,961	1,408
Total Instances	48,989	29,051
Daemon Virtual Memory Size Linux	48472 bytes	0 bytes
Daemon RAM size Linux	4868 Kilobytes	0 bytes
Daemon RAM size Windows (taskman)	4,2 MB	0 bytes

Table 3.1: Base resource requirements

so time is taken for UDP message transmission. In the figure 3.5b, the values for SNMP CPU monitoring spiked to as high as 15% in the application but most of the time the values averaged around 2-5%.

In the LNMP implementation, as with the SNMP CPU profile, we have taken the runnable portion of the process and detailed it here. As with both, parsing of strings takes a considerable amount of time of the total running time (15%). There is noticeably less CPU activity dedicated to similar processes when compared to the LNMP. This was most likely due to the printing out the values directly to the terminal. The heap size was also a bit smaller than that of the SNMP process.

3.2.4 Resource requirements

Table 3.1 illustrates the difference in values between the two different solutions. All unused/unnecessary libraries were excluded for these tests to ensure as lean a possible solution.

3.3 Conclusion

It can be seen from the experimental results presented in this paper that the newly proposed LNMP protocol retrieve similar results to SNMP, which demonstrates its accuracy but also shows itself as a good candidate for monitoring solutions. Granted, the system is not a drop in replacement for SNMP and all the network administration functionalities that has. But this is not what we were looking anyway.

It can be seen that the results are almost massaged to drive home the point that the solution is more lightweight. In some cases we are talking about a few megabytes of data here and there as well as minor advantages in computation time which are more leaning towards LNMP being advantageous. In a conventional workstation or desktop environment this would largely be considered a futile exercise, trying to preserve those extra execution cycles. This however, can not be the attitude adopted when managing resource constrained devices.

One needs to be frugal about what resources usage. We can see by measuring the RAM and CPU usage from the system we can see that LNMP proves to have the definite advantage. SNMP consumes too much resources processing community strings, which in turn translates to a larger system footprint. Similar to the uncertainty principle here, the more information we want to know about a system's status the more the monitoring solution impacts the system and becomes dependent on free system resources. In our case we want to consume as little of that as possible.

The solution, also by only recording values at a much more infrequent time interval as well as when the system status changes noticeably, provides a way for network administrators to ensure that their networks remain as unburdened as possible by the solution.

On its own, the fact that resource monitoring on lightweight networks via the LNMP proves to be slightly more lightweight is great. A version of SNMP could also probably be created to just include the features required for our monitoring requirements. LNMP does not even have to use SIGAR as its means of collecting system data. It could even do this with SNMP as a backend resource collection agent. (Albeit a slightly more bloated solution.)

That is the real benefit from providing this solution. When coupling its lightweight impact on system resources with its impact on network traffic data, the picture changes. The point here, is the more optimised methods of gathering performance data provided by LNMP far outweigh SNMP in its current form.

LNMP though is not a panacea for network administration solutions. Currently the software will not work on a lot of routers as these usually come with some proprietary kernel which is only configured to allow SNMP at best. More efforts could be invested in expanding the current solution to include both local (SIGAR) and remote monitoring (SNMP) capabilities in the project.

Combining this with the agent based monitoring infrastructure proposed by this project, you can have a solution that greatly alleviates bottle necks in large IT networks. Where LNMP enabled nodes are used to retrieve data from their surrounding nodes which only support SNMP, thereby removing the need for the central monitoring station as all the data is distributed through the network.

Whether the system resources gathered by the proposed solution is sufficient enough, remains to be seen. From the experience of working with the solution so far it would seem that is the case. In all likelihood, the metrics used to determine performance load on the network proposed by this protocol will have to be examined in detail and perhaps extended in a way which provides much more insight to a systems performance capabilities.

Chapter 4

Distributed Databases

Besides the need for a lightweight networking monitoring system, a cloud computing architecture requires a lightweight distributed database for storing and sharing data over the grid infrastructure. The purpose of this chapter is to display the performance differences between BigCouch and MySQL on the Alix system boards and a desktop PC. The goal here was not to use the fastest database technology but rather one more suited to the constraints or design decisions made in the project. While these constraints would not necessarily all be applicable to a global application of the software solution, in our focus we are working specifically with low power hardware and as such the constraints are specific.

Abbreviations:

NoSQL: Not Only SQL

JSON: JavaScript Object Notation

fstab: A file in Linux where all the partitions that the system needs to use are defined

DB: Database

TCP: Transmission Control Protocol

NAND: A Tyoe of flash memory

chroot: An environment that is used within the current operating system to create a separate virtual operating system

SQL:	Structured Query Language
MySQL:	A popular open source database management software solution
SQLite:	A very lightweight database library that can even run almost anywhere
WPA:	Wi-Fi Protected Access
AES:	Advanced Encryption Standard
VM:	Virtual Machine
PMI:	MySQL insert on persistent storage
PMR:	MySQL read from persistent storage
IMI:	MySQL insert to in-memory storage
IMR:	MySQL read from in-memory storage
PCI:	BigCouch insert to persistent storage
PCR:	BigCouch read from persistent storage
ICI:	BigCouch insert from in-memory storage
ICR:	BigCouch read from in-memory storage
PCM:	MySQL PC
PCC:	BigCouch PC
PC-MR:	MySQL read on PC
PC-MI:	MySQL insert on PC
PC-CR:	BigCouch read on PC
PC-CI:	BigCouch insert on PC
PCCNQWR:	BigCouch cloud insert insert n,q,w,r = 1

4.1 Introduction:

While exploring the greater solution of providing services in low power environments, as well as intelligence about the network environment, the need to monitor and store performance data in the network became more apparent. While initially the decision to mimic mobile phones in their decision to use SQLite as the database technology with which to store data for and about their operation, it was investigated upon that the technology was insufficiently capable for executing our requirements.

The specific constraints placed upon the solution by our unique operation on low power hardware and the inconstant nature of wireless itself, let alone long range wireless in low power mesh networks left a lot to be desired. Simply choosing the common database technology was not an option as much needed to be considered.

Our requirements for our database technology were the following:

- Low RAM usage
- Low CPU usage
- Small Binary size
- Distributed
- Not self contained (not restricted to being wholly contained on one host)
- Has redundancy features
- Fast (relatively fast, while understanding that a distributed database would not be able to provide speed on the same level as a local database such as MySQL, the slow down should not be noticeable that it causes human visible delays in operation)

We required low RAM and low CPU usage from the database solution because of the limited resources available to us on the Alix system boards. Ideally the solution would be a small package as well, with the binary itself being small enough to be easily contained within the available set of working memory. Typically a few hours after powering up the Alix system board, the GNU/Linux command 'free' displays that there are around 16MB of free RAM available.

The data being stored in the database is to be distributed and redundant so that if a node leaves the network, data can still be easily accessed from the available nodes with no reconfiguration being required. Seeing as this solution would exists in a wireless mesh network, this constraint would be essential for smooth operation in the network.

The requirement that the database not be self contained was decided upon the fact that Alix systems boards have very limited storage capacity. If we were to use MySQL circular replication in the network, it would mean that all the data being generated and stored in our database would be equally copied throughout the network. This would mean that for every N nodes in the network, N copies of the data would exists within the network, each wholly contained in each node; the database size would be N * R, where R is the number of records stored for by the host. This not only a waste of space in lieu of better storage options, but as the network increases in size, the cost of sharing this data would also increase.

Every insert, edit and delete operation would have to be synced across the entire network. If additional nodes are added, additional storage space would need to be added to every existing node on the network should the user wish to prevent the storage space from filling up faster. This would be more acceptable on a desktop or server solutionin cases where 1:1 replication is necessary, but as we are dealing with very limited secondary storage, that is not very fast in the first place, this is an unacceptable trait that cannot be considered in our selected database solution)

On highly available systems it proves almost impossible to maintain ACID properties while scaling across multiple machines when large datasets are involved (CAP Theorem). The only way to perform this is by using master-slave replication, which itself introduces a single point of failure. With MySQL replication one can not have more than one host set as a master[27]. This means that in introducing circular replication, you introduce a single point of failure. This is an unacceptable solution to a wireless mesh network.

This was were the start of the NoSQL movement began. Amazon's white paper on Dynamo, a highly available key-value store database[28]; spured a thousand NoSQL databases into existence. Since relational databases were severely limiting the ability to provide the always on experience, it became essential for databases to be re-thought.

It should be mentioned here that it was indicated that the project, although framed in a low power wireless sensor environment, has ambitions for broader adoption and usage. This means that the database being used was something that should be able to function on a scale much larger than a wireless sensor network. At the same time we wished to include all the functionality available in commercially available cloud computing environments such as Windows Azure. For this reason non-academic database solutions were investigated.

Relation databases	Table, Columns, Rows ACID properties fully satisfied Normalized to avoid data duplication Strong storage schema Queries fully supported
Distributed databases	Table like domain Data identified only by a key Schema-less Data integrity on applications code Eventual Consistency Support for queries is limited

Table 4.1: Common properties associated with relational distributed databases

4.2 Related Work:

Various distributed database solutions were examined to find ones with features that would allow for installation on the Alix system boards. Some of these systems are extremely complex and required more capacity than what would be available on the Alix system boards. As such, a database engine which runs with as little overheads as possible was desired. This decision or design constraint again threw out most NoSQL solutions. While some of the solutions appeared near perfect, it would usually come down to requiring implementing the technology in some memory expensive process, or VM to run.

It should be noted that this is but a small fraction of the available NoSQL distributed databases available. It would be impossible, considering our time constraints, to look at them all let alone test them. The data with which we assimilated this table was constructed from a previous study[27]. For a more complete list of NoSQL databases please refer to [29], currently 150 databases are listed.

Database Technology Name	Description	Shortcomings

Voldemort	Big, distributed, persistent, fault tolerant hashtable. Uses MySQL or BDB as backend for persistence on each node	Impossible to install a node to a live cluster. Entire system has to first be shut down. Requires load balancer nodes.
HBase	It is an Open Source, distributed, column-oriented store modeled after Googles BigTable [Chang et al. 2008]	Near perfect solution, albeit slightly complex in its configuration and large size.
Redis	Redis is an open source, BSD licensed, advanced key-value store	Master Slave relationship. Redundancy through replication not sharding.
Cassandra	It is a more complete key-value database based on Dynamoss fully distributed database design [DeCandia et al. 2007] and BigTables Column family based data model [Chang et al. 2008].	Bad/Non Existent documentation and very obscure and difficult API.
MongoDB	MongoDB is document-oriented approach for scalable distributed databases	Intricated cluster schema introduces several single points of failure. No full support for sharding and data replication constraints

Tokyo Cabinet/Tyrant	Tokyo Cabinet/Tyrant is an Open Source project claimed to be in use at mixi.jp, a japanese social network with 10000 updates/second through MemCache[27]	Does not have good documentation. Very few projects are using it.
Kyoto Cabinet/Tyrant	insert 1M records / 0.8 sec = 1,250,000 QPS, search 1M records / 0.7 sec = 1,428,571 QPS. Based on Tokyo Cabinet/Tyrant. No limit on database file	Poor documentation.
CouchDB	It has a totally unstructured schema-less storing backend throught the use of JSON format as data handling similar to Amazon's SimpleDB	Data needs to all fit on a single device. Redundancy through replication
Memcache	Open Source, high-performance, distributed memory object caching system	Does not have a persistence layer and requires underlying database technology for persistence.
BigCouch	BigCouch is in an elastic cluster, acting in concert to store and retrieve documents, index and serve views. Based on CouchdB	

Table 4.2: Table showcasing various distributed database technologies with their drawbacks in relation to the project

—

BigCouch was selected due to it being, extremely lightweight, and very simple to install. The solution itself did not provide much by the way of documentation and simply gave numerous instructions on how to configure the cluster storage of the database solution but not how to interact with the software solution at all.

This was due to the solution being based on CouchDB. A single instance of BigCouch, i.e. one node in the network, acts as a CouchDB node. So almost all the documentation applicable to CouchDB applied to BigCouch. There are some minor feature differences in BigCouch that arose due to the distributed nature of the project[30].

It should also be noted that no database drivers are provided with the software solution. As noting a common trend in access layers to database technologies, BigCouch and CouchDB moved to using HTTP as the transmission protocol and JSON for the encapsulation and serialisation of objects.

BigCouch is a document store database. This means as opposed to normal relational databases which have relations and tables, document store databases are oriented around documents and key-value pairs. This means that in one particular databases, each inserted record can be as different in structure and meaning as the previous. There is no predefined schema, and in the case of BigCouch, files can be attached to individual records.

MySQL was selected as the relational database with which we BigCouch will be compared against. MySQL was selected simply because it is the worlds most popular open source relational database[31] and with that all the optimizations and maturity that go with the solution. Something that most NoSQL solutions will not match.

4.3 Experiment Setup:

The idea for the experiment was to test the relative performance of the distributed database system and the conventional relational database technology. It was assumed that the experiments showing the performance of a distributed database would obviously be slower due to synchronous operations that would need to occur when information is being stored or retrieved over the network. Since BigCouch does not apply a memcache solution on top of its database infrastructure, this made all queries query the disk. MySQL on the other does apply very clever caching techniques.

The hardware being used for this experiment included an Alix systems board (alix2d2), a Raspberry Pi and a Pentium Dual Core. The alix2d2 ran a 500MHz AMD Geode LX800 with 256MB of DDR ram. The operating system is installed on a compact flash card and the disk runs in read only mode. On booting of the device, a virtual filesystem is loaded into memory to allow tasks to run that require read/write access to the filesystem. This mirrors the storage on the compact flash card.

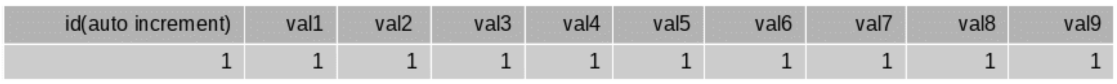
The Raspberry Pi features a 700 MHz ARM1176JZF-S, 512 MB of RAM. The operating system is installed on an SD card which is used both for booting and persistent storage. This was used purely as a reference and would not be used as a standard technology in the project, we were merely in possession of a unit and was able to include it for some basic tests. The results obtained from it will not be discussed or be part of any final decision made in the project.

The Pentium dual core was running an Intel E2200 CPU running at 2.20GHz, 2048 MB RAM, and an 72GB SATA 1 drive. This system was running Ubuntu 12.04. Gnome was removed in favour of LXDE as the former proved too resource intense. It should also be noted that while the Alix2d2 and the Raspberry Pi were both running in headless environments, the desktop PC ran with Xorg running.

All devices were connected on a private LAN via a TOTOLINK N100RE wireless N router. All the Alix system boards were connected wirelessly to the access point using WPA2 AES encryption using 802.11g. The desktop PC was connected with a standard CAT 5e cable to the router operating at 100BASE-TX.

A series of tests would be performed on each device to determine the speed at which it is able to perform read and write tasks to the database. The tests would also measure the state of the machine under load when performing these speed tests.

It was determined that 10 000 write operations would be measured on each



id(auto increment)	val1	val2	val3	val4	val5	val6	val7	val8	val9
1	1	1	1	1	1	1	1	1	1

Table 4.3: Table schema for MySQL data insertions

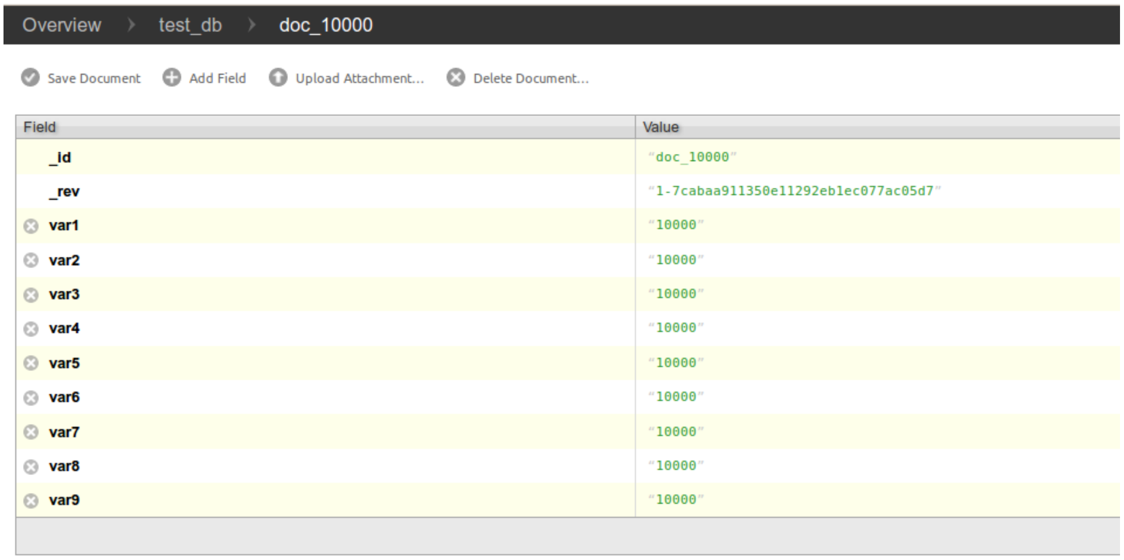
platform along with 10 000 read operations. The system performance measurements would be made with the GNU/Linux tools *time* and *vmstat*.

time would be used to measure the time it takes for the entire insert and read procedure to take place. The command outputs three different times on the completion of the task, namely real, user, sys. Real is the amount of time that has passed according the the wall clock. User indicates the amount of time the CPU has spent executing code in user-mode. 'sys' indicates the amount of time the CPU has spent executing code in the kernel space. 'user' and 'sys' indicate the total amount of time that the CPU has spent executing the process. Hence if multiple processes or threads are being used then this value is cumulative. This is not an issue since we are only executing single threaded inserts on single CPU devices.

According to the *vmstat* man page, it reports information about processes, memory, paging, block IO, traps and CPU activity. And when the command is issued for the first time since reboot, the output will be the averages from boot. Subsequent issues of the command will be based on a sampling period of a specified delay. For the purpose of this experiment a delay of 1 second was chosen to wait between capturing system performance snapshots.

Initially the idea was for each row to have random data inserted into each row, but it was made clear in a previous work[17] that the use of the random number generator was quite an expensive process. Using this on an Alix system board would end up more being an exercise for the random number generator than anything else. It was then favoured that incrementing integers would be used as the data inserted into each column value for each row.

Table 4.3 displays the base database table schema that will be used for the insert and read test operations. Where the values *val1* to *val9* would all be the same integer, with the value increasing in each row. On completion we should have 10 000 rows in the database with the final row having *val1* through *val9* containing



Field	Value
_id	"doc_10000"
_rev	"1-7cabaa911350e11292eb1ec077ac05d7"
var1	"10000"
var2	"10000"
var3	"10000"
var4	"10000"
var5	"10000"
var6	"10000"
var7	"10000"
var8	"10000"
var9	"10000"

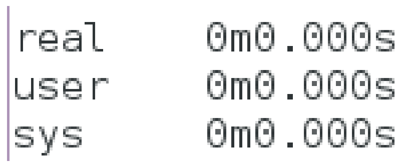
Figure 4.1: A record seen via the BigCouch web interface examining the contents of a single document

the value 10 000 in each column.

We create a similar schema for our BigCouch database. By schema here though we simply mean the general structure we will follow when constructing each document. This structure is not formally defined in the database technology, and only exists in each individual record inserted and in our insert and retrieve code. Checks have to be performed here to ensure that the correct data is being inserted or retrieved.

Each BigCouch database can be configured with the following performance parameters:

- N The number of copies to make of each document when storing it in the database.
- Q The number of partitions to divide the database into.
- W The number of nodes that must confirm successful write of the data before the write is acceped. If this is omitted the majority of nodes in N is used as the default value.
- R The number of nodes that must confirm they have the same result when making a query for a particular value before that value can be returned. The



```
real    0m0.000s
user    0m0.000s
sys     0m0.000s
```

Figure 4.2: Sample time output when running an application with the time command

default value if this is ommited defaults to the majority of N nodes.

The scripts used to insert and retrieve data from the database are available in appendix D2 for closer inspection

The process of insertion and delete are fairly simply and straight forward for both MySQL and BigCouch. Each record is inserted and queried sequentially to ensure a fair comparison and a large enough data set is used to average out query times(insert) over the course of database growth.

While the number of records used does not by any means constitute a large amount of data It does provide a large enough set for the Alix board which operates in a much more diminished capacity when compared with current server hardware.

vmstat -n 1 —awk 'now=strftime("%Y-%m-%d %T "); print now $0' > /tmp/vmstatlog.txt

The command issued above is to store the monitored system statistics. Stats are gathered every second, a date time is prepended to each gathered value which is subsequently piped into a file (vmstatlog.txt). This file is then used for analysing the system performance during the insert and retrieval procedure.

Each script is then executed by having the time command issued in front of the script command.

> *time bash couchinsert.bs*

With the results of this command returning output which is visible in figure 4.2

Before any of the scripts could be executed the databases had to be created on the various devices first. Since the Alix board was installed with a readonly file system it was in our interests to test this in comparison with an Alix board with persistent disk storage. It should be noted that before any experiments were done with either database technology, the other was disabled to prevent any interfering or skewing of the results. Each system was also rebooted before each subsequent test.

It was tricky getting the information into a read only filesystem. As we did not want to perform the writes directly to the files system, writes were performed to the in memory file system. The results then copied directly over the the persistent storage and then written to the compact flash card. It was chosen that this would be the better option instead of performing 10 000 write operations to disk which would most likely severely reduce the lifespan of the disk as compact flash cards have a very limited write lifespan.

The tests performed are also here to indicate relative performance of each technology, and while not wanted to compare apples to oranges, it was decided that BigCouch would operate in stand alone mode. This means that it operates as if it is a single CouchDB installation. No replication or distributed query operations are performed except where explicitly stated.

4.3.1 Using Persistent Storage

A USB storage disk drive attached to the Alix system boards in the network, was installed to act as a persistent storage area for the device. This was a Transcend JetFlash 600, which provides the fastest possible write/read speeds for NAND technology via USB 2.0. [32]. The flash memory device would also act as the storage location for files related to the databases software. That means, both MySQL and BigCouch would have to be reconfigured to perform all read and write operations from the new data directories.

While normally the procedure of moving data directories is fairly simple and can be done with simple symlinks, the Alix system boards being used in this project are running 'debian-for-alix' which makes this procedure a bit more complex. All configurations must be made within a 'chrooted' environment which gives read-write access to the underlying file system which is being stored on the compact flash card.

Once chrooted, 'fstab' can be accessed and persistent changes made. We then proceed to create symlinks from the various data directories to folders on the USB disk where the various database files are being stored.

Since ‘debian-for-alix’ mount’s the removable drives before ‘tmpfs’ has been mounted. This means that when the applications running, using ‘tmpfs’ the relative paths to symlinks are all incorrect. To solve this we use the ‘rc.local’ scripts file, as this allows us to specify startup scripts to run after the system has completed its booting procedure, and manually script system changes to run.

We also disable MySQL and BigCouch automatic startup on system boot, as this would assume to use the incorrect paths. Instead we rather indicate in the ‘rc.local’ file that the database services only start up after the USB drive has been mounted.

4.3.2 Using ‘tmpfs’

While it is great to see the performance of MySQL and BigCouch using secondary storage, it is also interesting to compare those results with results gathered from the same experiments run using ‘tmpfs’ as its primary storage without using other means to provide persistence.

This means that typically on reboot all the data we stored in the ‘in-memory’ filesystem would be lost. The experiments were performed immediately after each other for both MySQL and BigCouch. MySQL and BigCouch were configured as per the installation manual and started up normally. By default they would be using the volatile storage paths as their default paths.

4.3.3 PC insertion

The experiments performed on the PC will be to compare relative performance of MySQL and BigCouch on the two hardware profiles. We will also be performing the replication tests on PC. This operation does not really test the hardware limitations, but more how quickly the software solution itself can sync and store data across the network. It is assumed here that the biggest cost of transaction here will be the network I/O.

4.3.4 Replication insertion

The replication performance test would operate with 4 Alix system boards connected and 1 desktop PC. All were configured to have the BigCouch connected and configured to use each other as additional nodes in the same BigCouch cluster.

In order to perform this we needed to ensure that the Avahi-daemon was installed on each machine and configured each one's host name accordingly. This is because in the BigCouch configuration, a simple requirement for joining the network is for all the hosts to share a secret and that each host added is added to connect via the same unique network address. So this can either be a static IP address or a resolvable hostname.

It was deemed a much simpler solution to have the hosts operate in a dynamic network environment, as this is much closer to what the reality of a field situation would look like. In the 'hosts' file on each of the nodes in the network the line with the address for localhost is modified. Usually it reads as follows:

127.0.0.1 localhost <hostname>

Where <hostname>is the actual hostname of the machine. We now simply add another value immediately after <hostname>and separated by a space, <hostname.local>. This allows the Avahi daemon to pick up which hosts in our network we wish to connect to each other. Two reasons for using Avahi for DNS resolution are, the router itself would not resolve local addresses and Avahi works well in wireless mesh networks. So its great for discovering new devices on the network, meaning minimal configuration has to be done on the actual network itself when adding new nodes.

BigCouch would be configured and tested in the following configurations:

1. **r-insert (Replication insert):** Here we would create a database with fairly standard parameters: N = 3, Q= 32

2. **noq-insert (Noquorum insert):** Here we would create a database with the aim of having fast write and read operations with low reliability in ensuring the data stored and retrieved are correct: N = 3, Q = 32, R = 1, W = 1

	MySQL	Real	User	System	BigCouch	Real	User	System
Alix System Board	insert	637.17	475.578	145.661	insert	990.636	234.291	143.887
	read	641.426	464.237	151.961	read	734.772	224.822	151.821
Alix System Board	insert	647.106	473.982	146.745	insert	1732.761	245.735	151.925
	read	642.495	464.013	152.474	read	643.416	222.762	152.902
Desktop PC	insert	384.625	22.209	29.346	si-insert	901.204	55.375	53.287
	read	62.099	19.909	23.749	si-read	126.976	40.815	43.455
Raspberry PI	insert	1197.214	341.85	195.08				
	read	691.756	352.94	178.01				
Grid System					r-insert	8259.287	51.007	55.859
					noq-insert	1694.376	56.944	61.664
					replication read	337.313	54.855	58.812

Table 4.4: Table showing time values recorded when performing insert and read experiments measured in seconds

3. **rwnq-insert:** Here we will create a database that will be accessible via the network, but each document created will only be stored on one host. So, providing no redundancy and offering even less reliability than the query above.

4.4 Results

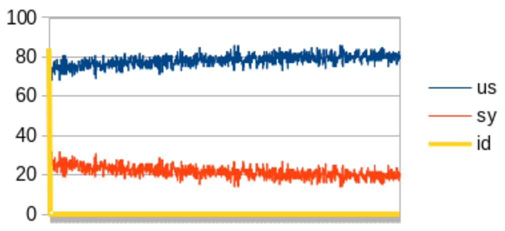
The first set of results displayed are tabulated results displaying the time taken for the experiments to complete. The results are grouped into different groups.

The first will be the CPU usage results of read and write tests performed on MySQL and BigCouch on the Alix Board with the in memory file system. This will be followed up with the test results using NAND type flash memory as the persistent storage area. The next set of results will be that gathered from performing the tests on the desktop PC and the tests performed to note comparatively the distributed storage and retrieval of data.

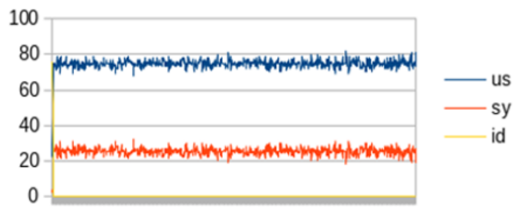
Following this the free RAM of the various devices will be displayed both before and during the tests.

Average MySQL query time across all devices	0.061298638
Average BigCouch query time across all devices	0.171341567
Fastest average MySQL write query time = (PC)	0.0384625
Fastest average MySQL read query time = (PC)	0.0062099
Slowest average MySQL write query = (Raspberry PI)	0.1197214
Slowest average MySQL read query = (Raspberry PI)	0.0691756
Fastest average BigCouch write query = (PC)	0.0901204
Fastest average BigCouch read query = (PC)	0.0126976
Slowest average BigCouch write query = (Replication insert)	0.859287
Slowest average BigCouch read query = (Alix tmpfs)	0.0734772

Table 4.5: Table showing time values recorded when performing insert and read experiments measured in seconds



(a) BigCouch tmpfs insert Alix

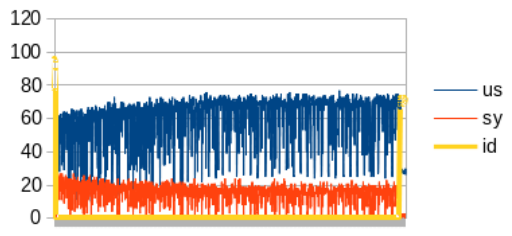


(b) BigCouch tmpfs read Alix

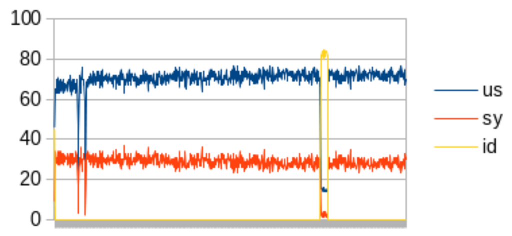
4.4.1 CPU usage

Table 4.4 contains a list of the DB performance readings. As can be seen from the table, MySQL outperforms BigCouch on the Alix system boards in terms of real world time. But it can clearly be seen that BigCouch spends way less time in user space than MySQL and spends about the same amount of time in kernel space as MySQL. It is assumed that the extra wall clock time taken by BigCouch can also be attributed to it communicating via a TCP socket to a webserver which returns JSON. So even though we are running this node in stand alone mode, a great deal of time is spent on I/O operations and waiting.

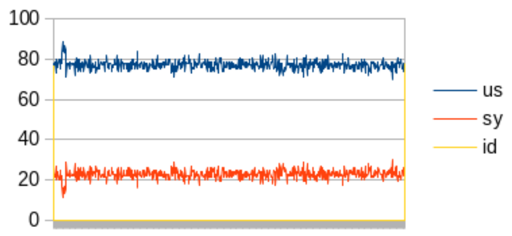
MySQL on the other hand is communicating via the local socket file instead of using networked communication. We could have changed it up to force MySQL to use the local connection, but this is not the default mode of operation for most MySQL installation in stand alone environments.



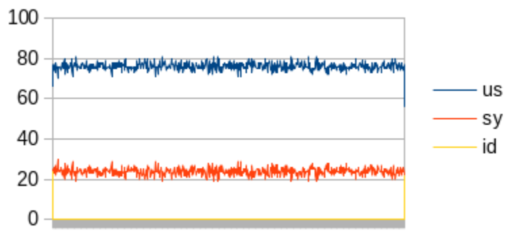
(c) BigCouch NAND insert Alix



(d) BigCouch NAND read Alix



(e) MySQL tmpfs insert Alix

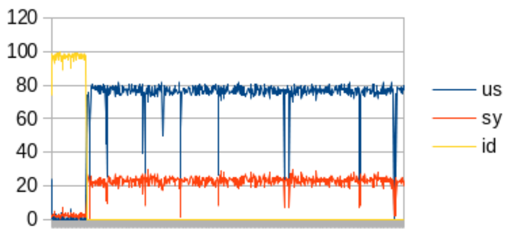


(f) MySQL tmpfs read Alix

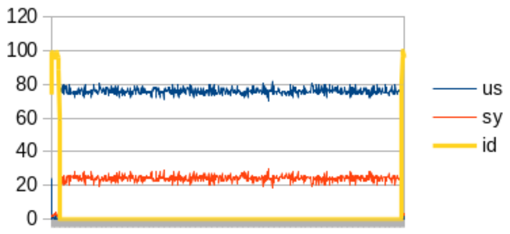
Also MySQL shows itself here as a very mature solution where years of research and development have been put into the product. It offers basically the exact same performance using the tmpfs and the NAND based disc storage. It is assumed here that some clever pre-caching techniques are being used here by MySQL. When previous work[17] was conducted experimenting with MySQL and TahoeLAFS, it was noted that MySQL is continuously interacting with the database. This could probably allude to the background optimisation that was going on ensuring queries are quick and responsive.

Upon further investigation it was noted that the MyISAM storage engine was being used. According to the MySQL manual, the AUTO_INCREMENT column is updated by MySQL on insert and update operations making it least 10% faster (a reference to a baseline performance metric was not specified). MyISAM also employs a cache mechanism to keep the most frequently accessed table blocks in memory[33].

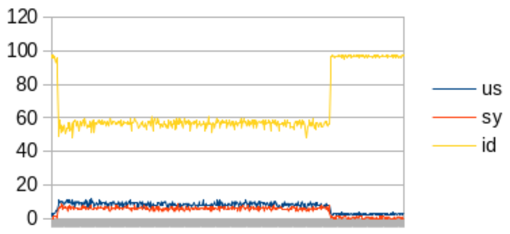
The BigCouch solution on the other hand seems to be suffering more from being a new solution with less applications using the solution and therefore less research being made available for the solution. With the times of using the NAND based



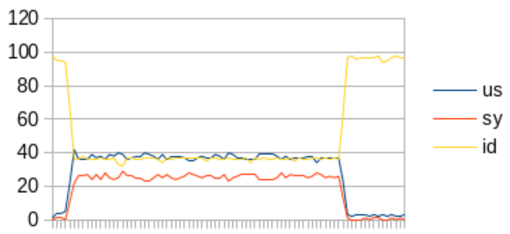
(g) MySQL NAND insert Alix



(h) MySQL NAND read Alix



(i) MySQL PC insert

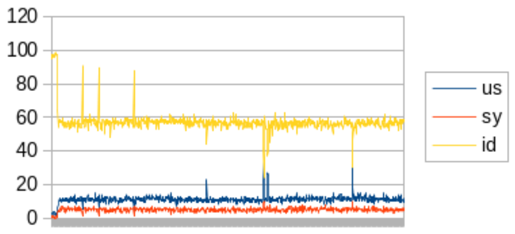


(j) MySQL PC read

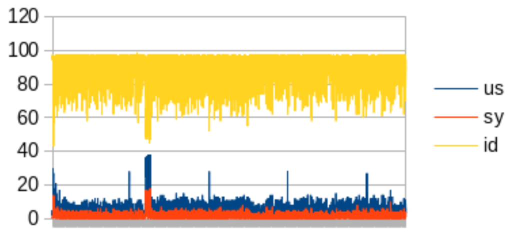
storage being roughly double that of the tmpfs solution. So on top of having to access the data via a webserver, the database engine now also has to query the disk which carries its own I/O burden.

As of yet BigCouch does not offer the same sort of performance and caching speed ups that MySQL has. The solution seems to have focused more on developing its B-tree storage engine[10] which focuses on maintaining efficient access operations (O(log N)). Alternate means of speeding up the performance of BigCouch will have to be examined but for now that will be considered out of scope of the current work.

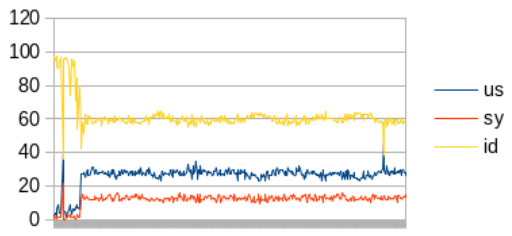
Unsurprisingly both BigCouch and MySQL perform noticeably better on the desktop PC than on the Alix system board. This is mostly likely attributed to



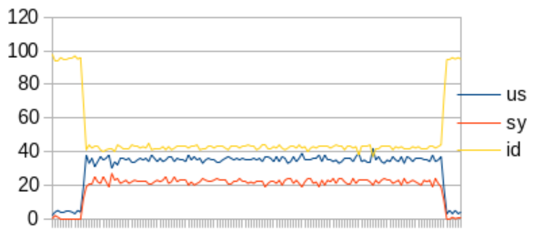
(k) BigCouch Standalone PC insert



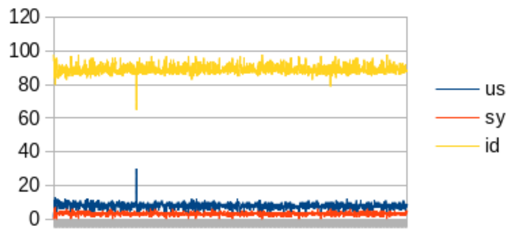
(l) Bigcouch r-insert insert



(m) MySQL PC insert



(n) MySQL PC read



(o) BigCouch Standalone PC insert

faster processing and bus speeds, larger and faster RAM, and faster secondary disk storage access. It is interesting to note the BigCouch inserts on the PC are as fast as the BigCouch inserts using the tmpfs on the Alix board. While noting the much lower user and system times, it is also another indicator here that either waiting on I/O or some other process within the BigCouch architecture is slowing this down.

It is slightly alarming to note how much longer it takes when data is inserted into the database using the default parameters when it is running in distributed mode. 8259 seconds is almost 138 minutes and most certainly won't suit high performance applications. Albeit in this instance we were running with predominantly low power hardware; our application, like most applications in this sort of setup, would preclude a high performance requirement. The system and user time also indicate very low values for storage, which leads us to conclude that an incredible amount of time is taken up by syncing and ensuring validity of the write.

The noq-insert yields results similar in terms of wall clock time to the Alix NAND inserts even though the user and system times are comparatively similar to the r-insert values. Having to not wait on confirmation of all participating nodes regarding the validity of documents being stored greatly improves performance. The replication read time is double that of single instance read times. This is attributed to having to wait for network I/O to complete before the query can be completed, documents can be stored on different hosts.

Looking at the performance graphs of the CPUs on the various platforms and tests we see that BigCouch performs nominally better when using the NAND based storage. Looking at this result and noting that the slowest read time took place when using ‘tmpfs’ it is assumed here that due to the limited free RAM being available, that some unused memory must have been swapped out at this time in order to make room available for newer processes.

MySQL performs comparatively the same across both file systems in terms of CPU usage, but we could have already inferred this result due to the similar time durations taken for the experiments with MySQL.

On PC we see both database technologies occupying very little CPU time, where the BigCouch stand alone inserts and reads perform comparatively the same according to CPU impact as MySQL does on the same hardware profile. Although MySQL does perform its read operation in approximately half the time as BigCouch and its writes in less than half the time. As we compare the performance of the machine though we can see it really is not struggling to complete the task at all.

When comparing these results to the distributed/grid inserts it becomes apparent that the host CPU is doing no work and just waiting. With the CPU idling for almost 100% and 80% of the time when performing r-insert and noq-insert respectively.

4.4.2 RAM usage

When looking at the RAM usage on the machines over the different tests, it shows a slightly different picture. Off the bat we can see that MySQL requires a greater amount of starting memory in order to function. These systems were stripped of all extra software solutions that were not required for the running of the developed software solutions.

We should note that in looking at table 4, that when comparing the free memory available on desktop PC systems with the MySQL service installed VS the BigCouch service that there also a noticeable difference. This change to a system just by having

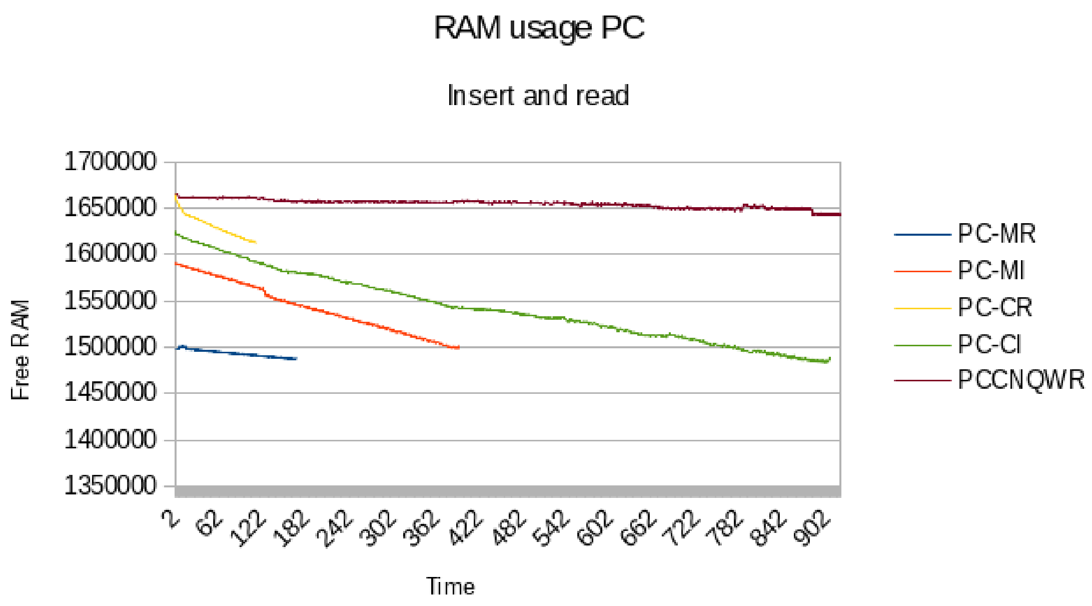


Figure 4.3: PC RAM usage

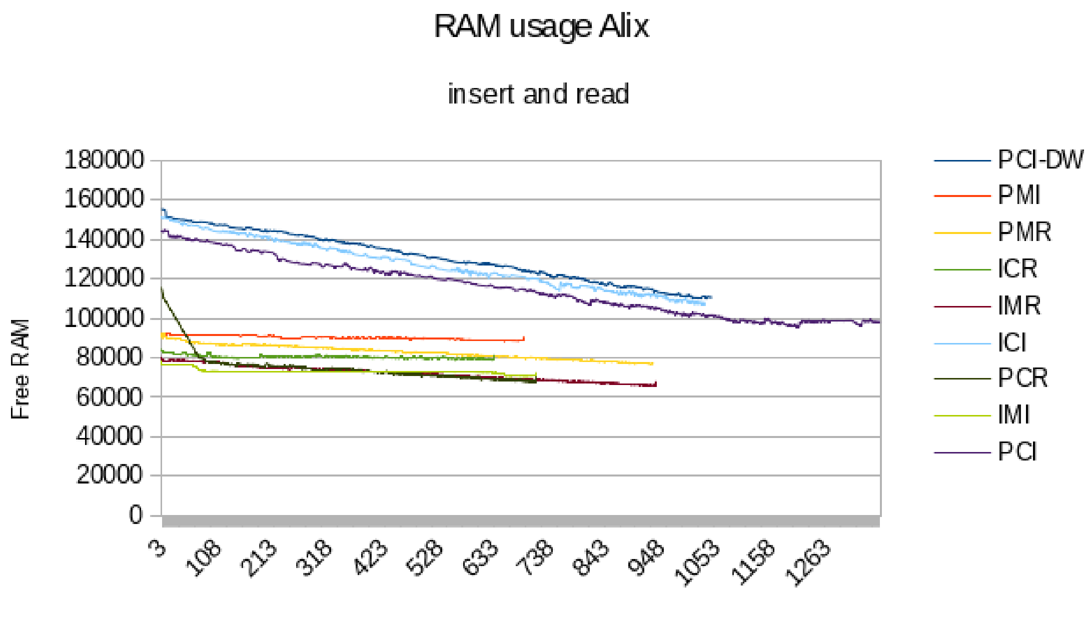


Figure 4.4: Alix system board RAM usage

a process loaded into memory was not made apparent by the previous CPU tests.

This could be the way Linux manages the memory of libraries and files. This in combination that all the running programs on the Alix board are brought into memory, it would make sense that the MySQL service would be more costly upfront to load, when referring to memory.

The only requirement required for running BigCouch in memory, is its HTTP server[34]. Upon examining the source code[35], we can see that it has been implemented in Erlang. And from the Erlang documentation[36], we can see that an Erlang process is very light when compared to operating system threads and processes. Thus it should not a surprise to note that the two systems consume very different amounts of memory. This more noticeable on the Alix Board simply due to the strict memory constraints placed upon the system and also not having any swap memory available to which could be used as cache for any running processes.

We can see that although MySQL has a greater cost to get the service loaded into memory, during operation it holds a fairly constant memory. Over the course of the operations MySQL seems to typically consume around 5 to 10 megabytes.

BigCouch, however, does not yield similar results. The system appears to use much more memory when compared to MySQL. Although the system still seems to get away with using less memory in total than what MySQL does. The only case where this is not the case is where we are reading from the database that is using RAM as its storage space. It is understandable that this uses considerably more space.

More research reveals that because BigCouch is a document store database it does not optimise the raw storage and retrieval of documents. It is up to the user to create views. Views are means for users to access specific data in a database. We can compare this to its SQL equivalent, a query. BigCouch optimizes these results by storing them in a tree. This tree is then updated whenever data is inserted into the database or queried, making the queries much faster. On first run, however, the tree must still be created; thus making the first run, i.e. our entire experiment sample to being this first run. (It is not entirely clear if the process of querying a document by its name/id has the same impact as querying a document based on a particular parameter)

In fact we can take it one step further and note that we did not use any views in our experiments. Meaning the results are never cached. Although it is not apparent as of time of writing this report if that amount of data is even cacheable or how BigCouch performs when querying the exact document vs querying a document based on parameters of that document.

We should also notice that on the desktop PC we get similar results showing that BigCouch occupies less system RAM in total as compared to MySQL. Its delta usage is higher than MySQL as we noticed before. We do note however, that the test PCCNQWR, which is a distributed BigCouch insert,(although in this case a single document is only ever stored on one particular node) that the system performance values remain pretty constant. This is in-line with the CPU performance values that we noted up earlier in the study.

4.5 Conclusion

MySQL should definitely be a first choice pick when deploying environments where performance is critical. Along with its popularity, its maturity has made it one of the most robust and polished solutions. It is arguable in the Open Source community whether the new ownership of Oracle can properly articulate the desires of the Open Source community at large. This is the reason why the MySQL project was forked and now MariaDB moves forward at least in the community as the forefront of Open Source relational databases.

MySQL, however, is not suited for installations which require high volumes of traffic and concurrency has it does not scale well horizontally. On large installations, the database becomes the bottleneck and distribution of data becomes a must.

In our experiments we have also shown that it might not necessarily be suited to environments where memory constraints are an issue. While in these experiments we show that BigCouch does consume more memory when conducting the tests, this is a stress test where we are reading all the available data and does not reflect the regular operation of the network, which will be considerably lower. (By design and on reflection on its desired implementation and environment)

BigCouch has also shown to occupy less operating system resources when waiting for requests. This means that more system memory will be available for additional processes on the machine. On more than one occasion the system has suffered from being unable to install or run packages because the available RAM had reached 0MB.

BigCouch does suffer from being significantly slower than MySQL and does not seem to contain all the performance tweaks that MySQL can boast about. Users accessing data might have to access it knowing that they might not be in possession of the latest copy of the particular document. But this seems largely ok. We don't seem to mind that our email is eventually consistent. A few refreshes of your gmail inbox are usually all that is required.

This measured slowness is not necessarily a bad thing in this particular context. It is not slow at the expense of system resources. It's perceived slowness in this case is actually its saving grace as it does not consume a lot of system resources when the system is loaded into memory. This is perfect for our low power environment and makes the solution highly suited because of its low system footprint.

Application developers just have to be aware of this issue when dealing with BigCouch, although it would be in our interests to conduct the experiments again on more enterprise grade hardware and make comparisons again to notice the difference.

It should also be noted that much more investigation is required into BigCouch performance tweaking. The solution is rather new and both MySQL and BigCouch installations were deployed with the default parameters configured in order to get the system up and running. A few performance tweaks were applied but not enough to consider it a thorough investigation. Not enough information is known about the solution in order to determine the most optimal performance settings.

At a later stage, we would like to re-conduct experiments using Tokyo/Kyoto tyrant. The authors claim magnificent database speeds. The only problem was the lack of documentation, which made its use difficult in this project as the time investment was too high.

Chapter 5

Lightweight Resource Allocation Protocol

The purpose of this document is to describe LRAP and its application. We go into detail of how the allocation protocol works and its place in the PCC ecosystem. LRAP provides a lightweight framework with which resources and services can be managed in wireless or wired networks. This particular application focuses on low power wireless mesh networks as that is the theme of the overall project. As described, one specific application is displayed but others have come to mind. The DNS server developed was chosen as it most readily displays the applicability of the framework. We show promising results with the DNS server being able to effectively manage requests and service dissemination throughout the network to allow for a more balanced approach to managing services in a network.

Abbreviations

LNMP: Lightweight Network Monitoring Protocol

LRAP: Lightweight Resource Allocation Protocol

UUID: Universal Unique Identifier

PCC: Participatory Cloud Computing

DNS: Domain Name Server

EC2: Amazon Elastic Compute Cloud

VOIP: Voice over IP

SIP: Session Initiation Protocol

5.1 Introduction

As with LNMP, we enthusiastically named this the Lightweight Resource Allocation Protocol. And, as with LNMP, we use lightweight here in the sense of the process itself being low on its resource requirements. LRAP is designed that services can be provisioned in the network while distributing any possible impact these services might have throughout the network.

While it is essential in network management to perform monitoring of the network to ensure that we are aware of changes in the network and respond to issues arising, we are ever in engaging in a world where our digital life is autonomous and requires less and less human interference in order for the operating and running of services.

This requires that our networks be carefully constructed. Huge cloud services such as Gmail and Amazon's EC2[37] do not have a team of engineers sitting behind desks, carefully watching systems logs and effecting changes in the networks themselves manually by analysing these results.

No, these changes are carried out in large by automated scripts and processes which monitor the state of the various systems involved. Then, either by predictive, heuristic or even static allocation processes, assign tasks to various nodes in the system in order to carry out the various tasks.

Human, interaction in these large datacenters are now left to rebuilding broken storage arrays or lean highly paid emergency response teams waiting for something to go wrong. Cloud computing system has now long evolved from the support desk agent sitting around waiting for a customer telephone call in order to set up a new web hosting environment. Now, one can spin up an Amazon EC2 instance in a matter of minutes.

So equally in this project where lightweight monitoring processes are in place, we need to also ensure that once our changes are detected in the network that we can in turn act upon those changes and make effective decisions in service allocation in our network based on these decisions. If we can make these sort of changes/decisions in our networks, we can ensure that service provision is somewhat attainable.

The problem of allocation needed to be solved in this part of the thesis was an optimization one. The problem that this most resembled was the greedy knapsack algorithm. We then modified the greedy knapsack algorithm to be more suited towards this problem space that we were dealing with.

With our modifications to this algorithm and also our implementation, we are able to easily connect existing services with our algorithm to solve service requests. Current resource allocation protocols for grid computing systems focus on computation tasks as this is usually the primary function of compute grids. The LRAP focuses on lightness and no complex calculations are performed when deciding where services are allocated.

Services in the network could be anything ranging from VoIP services such as Asterisk or servicing various web services/websites to users. Services, which are usually accessed in networks by various domain names.

These names are usually the addresses of nodes on a particular network or spanning multiple networks. These host names in turn usually reflect the names of the services that they are running should they be solely dedicated to serving a particular task in the network.

Servers though can have multiple domain aliases though depending if the machine will be used for multiple tasks on the network. This is useful because it allows us to query services without binding the service to a specific node in the network. Should we wish to move our printing service running on ‘print.local’ pointing to IP address ‘192.168.1.7’ to instead point to ‘192.168.1.8’, we can do so without reconfiguring all the other nodes on the network. We can simply reference the same hostname which has just been moved to a new host.

This portable service management is the idea behind this implementation of the LRAP. It makes sense in networks that when we refer to services we do so in a way which allows us to easily identify nodes and services in our networks. This makes the job as a network administrator much easier.

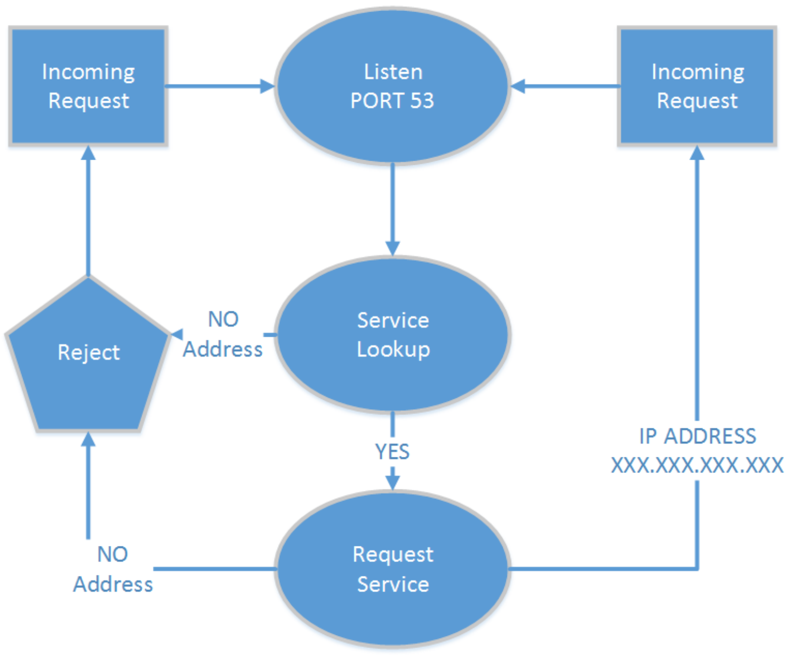


Figure 5.1: DNS request flow

In this instance the services are queried by the networked name; when we query our VoIP/Asterisk server, we would query 'asterisk.site' on the local network and have the DNS server resolve it. The prerequisite here being that each node would have to be preconfigured with the correct DNS server to resolve it.

In this case we are running our own DNS server which in turn connects to the LRAP agent. The incoming service name is analysed. The service name is extracted from the full domain name and all we are left with is 'asterisk'. This value is then piped into the LRAP agent which in turn searches through all the available hosts it has connected to it and only upon finding suitable candidate, will it return an IP address. It returns the equivalent of 'host not found' in the case that the service does not exist or the domain name was invalid. (It will only accept requests for services in the particular defined domain, in this case '.site'.) If the service does exist but there is no available host to accept the request then the service returns 'host rejected'.

The DNS server then reviews the response from the LRAP agent and returns the correct output to the program making the request. The process of querying the LRAP agent is totally transparent to the user application and it appears as an

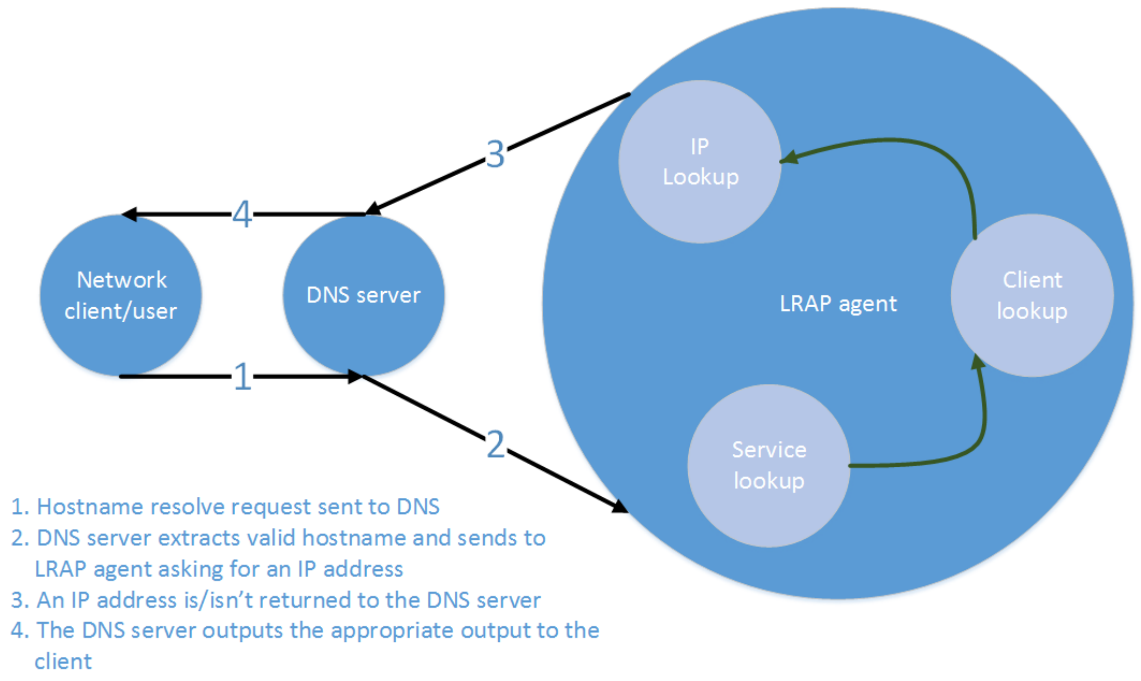

Network client/user
DNS server
LRAP agent
IP Lookup
Client lookup
Service lookup
1
2
3
4
1. Hostname resolve request sent to DNS
2. DNS server extracts valid hostname and sends to LRAP agent asking for an IP address
3. An IP address is/isn't returned to the DNS server
4. The DNS server outputs the appropriate output to the client

Figure 5.2: More detailed DNS flow

ordinary lookup has taken place. This is ideal since no changes to the way that networked software communicates has to change.

While an application in DNS with LRAP is one way to display the effectiveness of LRAP, it is not the only application that comes to mind. One could use it in a situation where you have a highly connected system of telephony switches. Switches are more advanced PABX (private branch exchanges) typically responsible for handling extremely large volumes of telephone calls.

Usually the approach in VoIP environments to manage call loads is by using round robin techniques via DNS or SIP proxies to split the calls to each server. This could be changed so that instead of using simple load balancing techniques, our more deterministic method could be applied. As VoIP is a service which the quality is very much affected by the underlying infrastructure, anything which can cause latency to rise above 200ms can introduce delay or even jitter in the call which makes communication more difficult.

This approach presents itself as a solution in which problems were resource allocation and load balancing can be optimized instead of simple load spreading techniques. Where load spreading is just the simple task of distributing tasks between execution points by some predefined ratio.

5.2 Experiment Setup

The purpose of this experiment is to test the applicability of LRAP in a low power wireless networked environment. It was decided that the easiest implementation thereof could be demonstrated through an application using DNS.

A DNS server is installed on one of the nodes in the network. All service requests made in the network will be done through this DNS server. This required that all services in the network be labelled appropriately. What we mean by this is that, if there is an Asterisk service on the network, then the URL required to access that service has to be labelled asterisk.<domain>.

By default the default service domain of the network is '.site' and will be used unless an alternative is provided in the configuration file. This means the way to fully specify a domain name to access a service on the network, in our case the asterisk service, is by its full domain name: 'asterisk.site'. Obviously one could access the service on a specific node directly if one knew the IP address of the node. This would be counter-intuitive to the experiment that we wish to conduct here.

The DNS server, after receiving the request will connect to the LRAP agent and request an IP address to provide to the connecting client. If it receives a 'FAILURE' from the allocation agent then it proceeds to inform the querying client:

ping: unknown host maps.site

On the other hand, if it does return a 'SUCCESS' and the correct IP address, then the IP is returned to the connecting client.

The selection of the library to use for implementing the DNS server was a bit tricky. Eventually it was settled on using 'evldns'. The package is described as being a combination of libevent's high speed event handling code with ldns' DNS packet manipulation. As indicated the library requires libevent (>=1.4.9) and ldns (>= 1.5.0).

The library proved to be extremely light and was relatively easy to install. What was really tricky was understanding the DNS protocol itself and understanding how to return meaningful results to user requests.

5.2.1 System installation:

An interesting observation was made when each node was completely reset from the previous installation. Since a new image was installed which had to account for the additional packages which needed to be installed on the machine, to facilitate the running of the LRAP system, the system state was altered and the nodes needed to be reinstalled one by one. This process was completed with each node being disconnected from the network one at a time, formatted, and then reinstalled.

After the process was complete the machines were examined and it was discovered that the databases had been restored to each individual machine. Since BigCouch stores each nodes' identity based on each username as well as the state of each host in the network on each node in the network, the databases are easily restored to an operative state when anomalies are detected.

Each host was disconnected for close to an hour, so it is not clear how long this process took and should be tested at a later stage, as it is not part of the current study. However, it is good to get a first hand look at how impressive the solution has been constructed and reliable the redundancy features are.

The process of getting each node prepared for the installation of the LRAP solution proved to be quite difficult. As the default 'debian-for-alix' solution only comes with a default partition size of 2GB. While this is fine for most deployments on these devices, since they will primarily be used as routers and small file sharing devices, it proved to be quite limiting for our project.

We had to thus first expand the local storage. But this was not a straight forward process either, as the tools to expand the partition require the device to formatted as ext2, while the 'debian-for-alix' image comes with ext3 as default. Thereafter the block device can be expanded. A filesystem check is performed and it is converted back to ext3. This process is detailed in appendix d1.

This process however, was only the start of our woes. The issue of expanding the filesystem only came up after repeatedly trying to install the base packages we required to run the LRAP prototype. After struggling to install the latest version of libboost available on debian squeeze. It was realised that this version (libboost1.43-all-dev) was too low and did not contain the correct feature set in order to run the solution properly. The system needed to be upgraded. Usually debian distributions prefer the complete re-installation in order to switch to the latest software packages. This is usually because the distribution runs to a point where the system is considered stable and no new packages are added to the repository except if they are security updates.

Sometimes boot processes change, such as moving from 'initrd' to 'systemd'. These are two completely different ways to manage system service and booting of the system. We however did not have this luxury of installing the system from scratch. Instead, the apt-sources were changed from squeeze to wheezy. The apt tree is then updated, and the installation of the latest boost libraries then proceeds (libboost1.49-all-dev). To note, this is the latest package available for wheezy. The latest release of build-essential is also required.

After the complete system was installed, tests were conducted to ensure that all the correct packages were in place, and any errors encountered were rectified. An image was taken of this completed system and used as the base flash for each of the other nodes in the network.

Each system was then just edited to ensure that each node had its own unique hostname and BigCouch identifying information as well as its own LRAP configuration. All of the hosts in the network were acting in client mode except alix1.local which was also the node selected to act as the DNS server.

5.2.2 Hosts in the network

The hosts taking part in this experiment are detailed in table 5.1 Each host has a defined set of services. These services are listed in table 5.2 along with their weights. The weights defined are approximations of what the system impact would be. Ideally these values would be computed and dynamically adjusted for each node type in the network depending on the particular impact of each service on the system.

We would also like to note that the total processing power(TPP) left for the so-

Hostname	System Type	CPU	RAM	IP Address
alix1.local	Alix system board	500MHz AMD Geode LX800	256 MB	192.168.1.7
alix2.local	Alix system board	500MHz AMD Geode LX801	256 MB	192.168.1.6
alix3.local	Alix system board	500MHz AMD Geode LX802	256 MB	192.168.1.8
alix4.local	Alix system board	500MHz AMD Geode LX803	256 MB	192.168.1.9
beige.local	PC	2.2GHz Intel Pentium Dual	2048 MB	192.168.1.3
cheddarcheese.local	PC	1.8GHz Intel Core 2	1024 MB	192.168.1.2

Table 5.1: Table showing the hosts part of the network

Name	Weight	Port	Service-URL
asterisk	3	5060	
stats	4	80	<host>/stats
storage	2	29999	
email	4	587	
maps	11	80	<host>/map
calendar	23	80	<host>/calendar
store	12	2342	
printing	6	8763	
webservice	4	8080	<host>/API/
webserver	8	80	<host>/

Table 5.2: Service list and associated weights

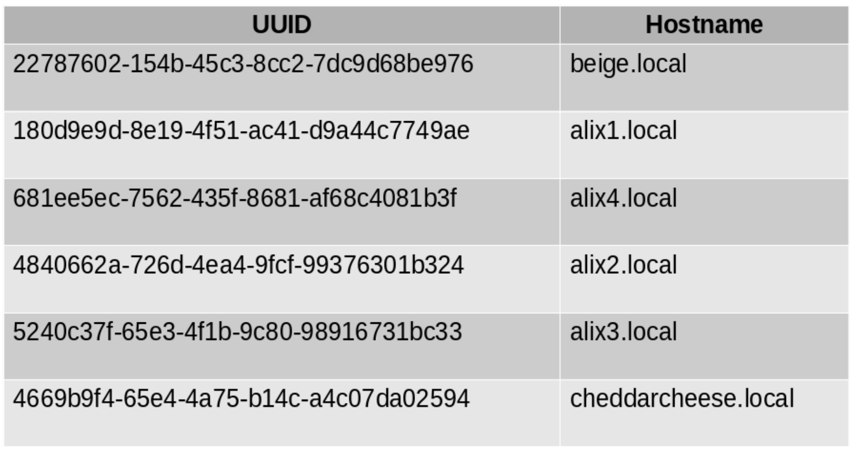
lution, is the defined upper limit which prevents new processes from being allocated to the host once this value has been reached. This value, as explained before, is realised by calculating the current load on the system and proportionally allocating a penalty to the host based on that load calculation.

Table 5.3 is a list of services available on the network and show which services are available on which hosts. It should also be noted that the desktop machines in the network have been allocated a greater TPP as their capacity to perform is much higher than the Alix boards connected in the network.

The following table maps the UUIDs for each to their host. The solution does not reference a host by its hostname or IP address. All information is managed

Host	Service 1	Service 2	Service 3	Service 4	Service 5	Service 6
alix1.local	asterisk	stats	storage	email	maps	calendar
alix2.local	store	printing	webservice	webserver	asterisk	stats
alix3.local	storage	email	maps	calendar	store	printing
alix4.local	webservice	webserver	asterisk	stats	storage	email
beige.local	maps	calendar	store	printing	webservice	webserver

Table 5.3: Hosts and their associated services



UUID	Hostname
22787602-154b-45c3-8cc2-7dc9d68be976	beige.local
180d9e9d-8e19-4f51-ac41-d9a44c7749ae	alix1.local
681ee5ec-7562-435f-8681-af68c4081b3f	alix4.local
4840662a-726d-4ea4-9fcf-99376301b324	alix2.local
5240c37f-65e3-4f1b-9c80-98916731bc33	alix3.local
4669b9f4-65e4-4a75-b14c-a4c07da02594	cheddarcheese.local

Table 5.4: UUIDs and the corresponding hostname associated with it.

by UUIDs in the network. This way, if IP addresses change on the network due to dynamic configuration, the system still maintains the same identifying information.

5.2.3 Experiments

DNS Tests

The experiments will be conducted to demonstrate both the limits of the system as well as its ability to reduce load on specific nodes in a network by spreading it around in the network. The demonstrability of the solution itself is inherent in these tests so showing the system running in its entirety as its own separate test becomes redundant and is omitted.

Stress Tests

These tests will be used to detect the state at which the system collapses. The results will be compared to that of the results achieved by the creator of 'evldns'.

The stress tests will be performed by slowly increasing the number of requests being performed on one node. The frequency will be adjusted at punctuated time intervals as to better monitor and notice changes on the nodes.

The system statistics will be recorded as before by using 'vmstat' to collect the system information every second. For this experiment an initial value was selected of 120 seconds was selected. This was representative of the amount of time that had to elapse before the frequency of requests is raised for the given time period. The frequency of requests was set to be raised by 20%. A large time window was selected to determine if there was any cumulative load effects burdened on the system by the running of the software solution.

In this experiment two hosts are selected. One will be the node where the LRAP service is being run, whilst the other will be the node making the requests to the server.

While this experiment will be conducted the following assumptions and parameters will be set/determined:

- Upon making a request (for these request ‘nslookup’) will be used to perform the hostname lookups.
- A service will be selected out of the list of available services. (see table 5.2)
- The host will not contain all these service so some services will not return IP addresses. This however is irrelevant as we are not testing the responsiveness of the server to specific requests.
- The list of available services should quickly expire as the services are being polled.
- Thereafter the node should return the equivalent of ‘host not authorized’ (’LDNS_RCODE_NOTAUTH’) error codes.
- The system will continue to perform scheduled monitoring of itself as well as the determination of free resources within the system and ordering of services.
- It should manage this while continually receiving an increasing amount of requests.
- The frequency of requests will start at 1 request per second and continue until it becomes clear from the visible output on the command line that the system is unable to cope with the load.
- These tests will be performed on the Alix system board(alix1.local).
- It is also understood that it might not be possible reach the maximum load on the machine due to the process of querying taking too long, while not necessarily putting load on the system. That is, the system is waiting for various

I/O operations to complete instead of the having the CPU active at the time computing the results.

The polling script was written in python, which will echo system commands. ‘nslookup’ will be called by the subprocess module[38]. This particular version is a backport from python 3.2 as it is guaranteed to be reliable in threaded applications. This is used because the calling method is spawned as a process and will not delay subsequent calls. So query requests per second can be ramped up without having to wait for the previously polled operation to complete.

This is to better ensure that the experiment is not waiting on any operations of the node being tested to complete as this might cause delays which might not necessarily be pushing the limits and give an inaccurate reading of what the system might actually be doing. This would be the case as discussed above where slow queries could halt the entire process. Using threads allows us to work around this and better control the queries being sent out.

Performance Tests

These tests will just be used to determine the overall operating values of the system within the network while the software solution is running and DNS requests are being made. The requests will be made at constant time intervals as opposed to the previous test. This is so we can better detect changes in the network and operating performance on the host nodes.

All performance values will be stored in the network using the existing BigCouch database infrastructure that was setup in the previous experiment. The idea here is to make requests in a similar fashion to the stress tests. This however is not a test for the systems’ limits but more to demonstrate the effects in a networked environment.

The experiment will be described as follows:

- The entire network is connected in its simulated deployed environment, where all the nodes are connected to each other wirelessly.

- The BigCouch database instances are running on all the connected nodes in the networks. So each instance will store its system data within the BigCouch cloud. This will allow for the allocation agent to monitor and calculate new performance values based on the load information stored over the previous minute.

- The DNS server is started on the dedicated DNS node (alix1.local) This is simply configured in the configuration file and the DNS server is started on boot. The correct permissions are required as port 53(DNS listen port) requires privileged access.

- A polling server (VirtualBox VM) is setup with python installed to run the testing script. As before, the subprocess module is used again as the Popen call that it provides is non blocking. This will allow us to easily send multiple packets to each hosts in the network in quick succession.

- The polling server is then selected to send payload packets to send service requests to each node in the network. This will be done with all the nodes being connected in the network. The polling server will make a request to resolve the hostname selected which will be chosen by randomly selecting from a list of available services.

- The ICMP protocol will then attempt to resolve the service with the present DNS server specified in '/etc/resolv.conf'. This will then in turn query the contact the DNS server which should resolve the received request to one of the service nodes on the network before the ping process occurs.

- The first attempt at sending a payload consisted of attempting to send various network attacks to the hosts on the network in an attempt to cause them to increase system resource consumption by processing the exploits of these attacks. It seems that with recent kernel updates over the years that most of these tools are now just simply unavailable to exploit. The only feasible way to recreate similar scenarios where systems fall over involve more than just the node in question. The various methods glanced over were SYN attacks, Nuke attacks (old), application level floods, ICMP floods to name a few.

Distributed attacks would be necessary to carry this out. This would meant that other hosts in the network would need to be added just to perform the attacks. Then synchronisation would also have to be established as well as taking into account the slow down that might occur. It also came to light that the performance hit, while in terms of access to resources would be fairly quick, we were looking for something that would on a more immediate impact the system.

To work around this, then Apache web service was installed on each machine in the network along with PHP5. The process of sending a payload was replaced with running an actual process on each machine. The idea here being when the polling server requests a service on the network, the DNS server will respond with the appropriate IP address. The polling server will then take the IP address returned by the DNS server and in turn query a PHP script via Apache2. Take a look at the example service request displayed in figure 5.3.

- The polling server will then access the PHP script. The type value supplied in the URL is in reference to the service being accessed on the machine. This is done by creating a bash/python script which uses curl to access the resource at the specified IP. We are able to retrieve this value with relative ease by using the 'host' command. This even allows us to specify the DNS server that we wish to query, without having to change the default nameserver on the system specified in '/etc/resolv.conf'.
- The PHP script will then proceed to execute a loading process while will consume a portion of the CPU time and memory representative of the weight that the process is associated with. The aim is to capture the overall system load averages across all systems.

Various techniques were investigated to have the system simulate load on each machine using PHP. The first investigation looked into performing floating point

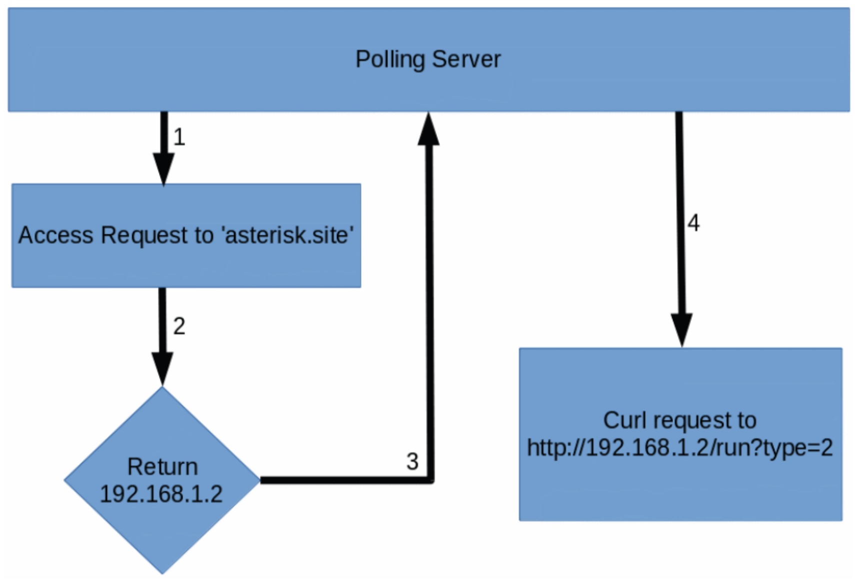

Polling Server
1
Access Request to 'asterisk.site'
2
Return 192.168.1.2
3
4
Curl request to http://192.168.1.2/run?type=2

Figure 5.3: The service request process

operations for a specified amount of time. These processes did not have lots of control about them and would differ from machine to machine. Instead the 'stress-ng' solution was opted for as it delivered far more granular control over the loading process. The original solution, 'stress' was developed Amos Waterland[39] at Harvard but was rewritten (Colin King[40]) to provide even more fine tuned control over the loading process.

5.3 Results

5.3.1 Stress tests

A frequency level raising of 20% was selected as it was determined that this would be adequate enough to determine where the point is when the system collapses. This initial setup though proved not to test the system at all. Requests were being made to the system every 0.00241785163923 seconds and the system processor levels were not even breaking past 2%.

These experiment values were thus tweaked again new values were selected. Instead of 120 seconds, a value of 20 seconds was selected to allow to pass. The previous experiment had taken quite a long time to ramp up its query times and still yielded no results. The frequency of requests was also changed from 20% to a 30% increase in polling requests.

It is assumed at this stage that the slowdown could have been partially affected by output the values being sent and received to stdin and stdout respectively. This information was also been sent over SSH. This could have resulted in skewed results as the output from the scripts to the terminal could have been delaying the iterations as the terminal starts reaching a limit to how fast data can be transmitted and printed to the screen. The script values were tweaked as mentioned above and all output to the command line was discarded and sent to '/dev/null' so as to not have the console and network I/O slow any of the apps down.

It was also determined that in our tests we were using a random number generator to select a particular service from the network which was causing the CPU to perform a lot more calculations per iteration. Random number calculations are expensive when you are making a query every 1ms. This would cause the CPU on the side of the testing machine to become more loaded than the DNS server. This was disabled in favour of one particular service being selected for in the DNS queries. In this case 'asterisk.site' was the hostname selected for query.

It was noticed that when increasing the amount of requests too quickly, we missed the point where the DNS server became burdened and we notice a trend of rising CPU load values, but suddenly this falls due to the polling server becoming too overburdened with the task of sending out queries to the network. It also came to light that the use of threads in the testing scripts was redundant and was causing overhead which prevented us from attaining better results.

The problem still remained though. The polling server would become overburdened way before the DNS server was indicating any level of load which could be considered high for the system. This process was then abandoned in favour of a bash script which would send the task to the background. This means that as soon as the command is executed, the program is moved to a 'background' state and the user gains access to the terminal without having to wait for the task to complete. Allowing us to execute another almost immediately after the command has been issued. This is possible in Linux by appending an ampersand after the 'nslookup' command. Thereafter the script would sleep for a calculated duration. This process of spawning tasks though still proved to be too resource intensive on the host machine.

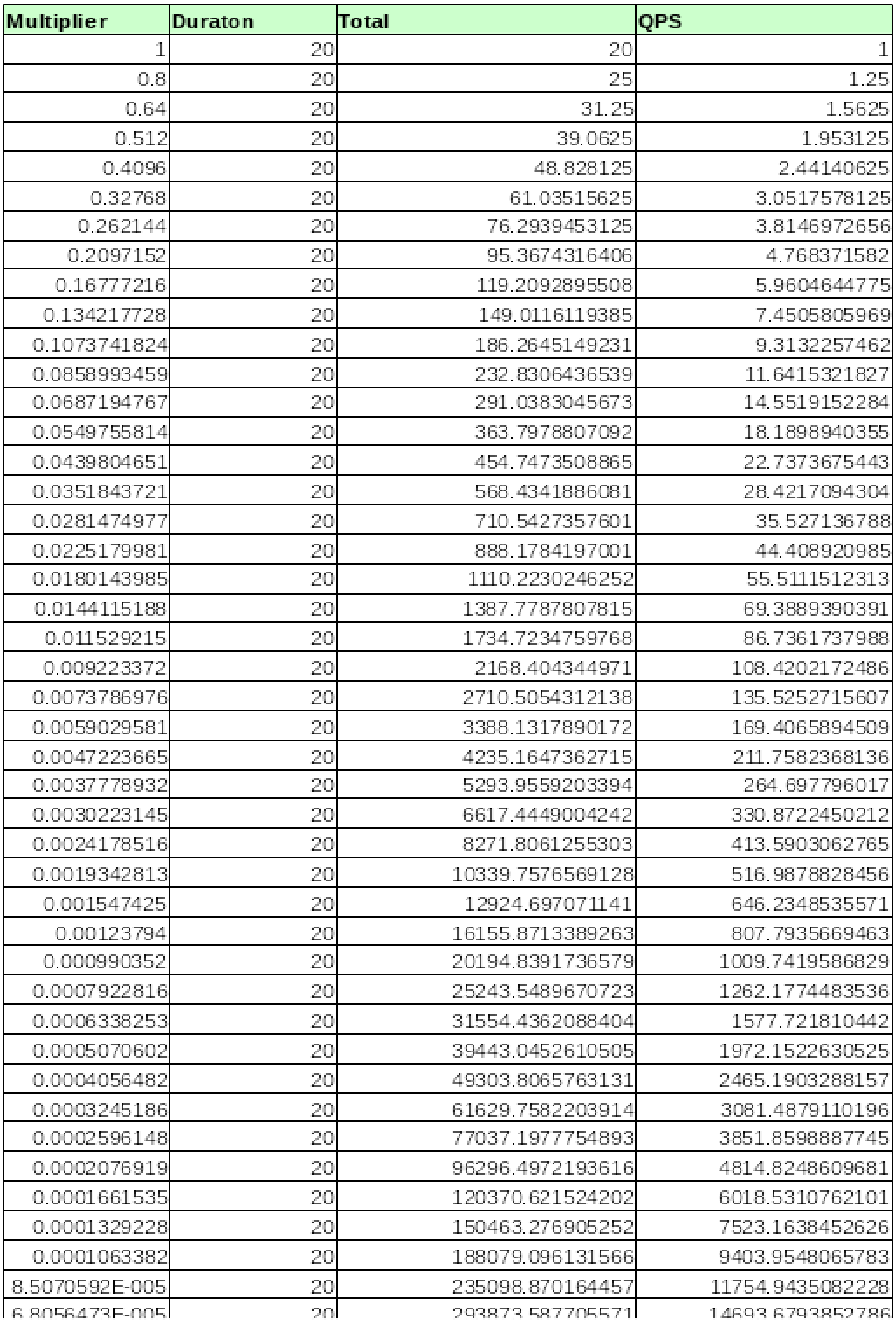
This process was again abandoned. The task of loading the DNS server with request was again approached this time via a distributed angle. We took 3 different nodes in the network (beige.local, cheddarcheese.local and an I7 HP envy notebook), and proceeded on all three nodes to begin to spam nslookup tasks. On the notebook this process was spawned multiple times as each script would be run on its own execution thread which in turn would occupy its own CPU thread.

This still proved to be a fruitless activity and external solutions were queried; spawning multiple 'nslookup' tasks was far more resources intensive than responding to the query performed by 'nslookup'. The solution provided by Frank Denis was downloaded and modified. Frank's solution(dnsblast)[41], bombards a DNS server with resolve requests. The solution had to be modified since the original was just spamming 4 random bytes and appended a '.com' suffix to whatever randomly generated URL was formed. This was changed to be modified to be an address which was appended with a '.site' suffix instead. This is because the DNS server does not perform a lookup to see if a URL exists if it does not end with '.site'. Thus performing considerably faster than validly suffixed requests.

The server does however perform the same amount of work when look and returning a valid URL ending in '.site' as well as an invalid site ending in '.site'. It is noted that 'dnsblast' is not a tool for DoS'ing resolvers, though in our case while our goal was to determine the moment where the system collapses(in essence a DoS attack), we needed to monitor its progress until that point. Thus making it the correct fit. We could monitor this point because we could adjust the number of queries to send as well as the frequency of the queries sent.

We modified our bash script and used the following values to ensure we have a constant time time measurement for each frequency level. The values can be inspected in table 4.

The multiplier values selected were as follows: The multiplier is a product of itself :



Multiplier	Duraton	Total	QPS
1	20	20	1
0.8	20	25	1.25
0.64	20	31.25	1.5625
0.512	20	39.0625	1.953125
0.4096	20	48.828125	2.44140625
0.32768	20	61.03515625	3.0517578125
0.262144	20	76.2939453125	3.8146972656
0.2097152	20	95.3674316406	4.768371582
0.16777216	20	119.2092895508	5.9604644775
0.134217728	20	149.0116119385	7.4505805969
0.1073741824	20	186.2645149231	9.3132257462
0.0858993459	20	232.8306436539	11.6415321827
0.0687194767	20	291.0383045673	14.5519152284
0.0549755814	20	363.7978807092	18.1898940355
0.0439804651	20	454.7473508865	22.7373675443
0.0351843721	20	568.4341886081	28.4217094304
0.0281474977	20	710.5427357601	35.527136788
0.0225179981	20	888.1784197001	44.408920985
0.0180143985	20	1110.2230246252	55.5111512313
0.0144115188	20	1387.7787807815	69.3889390391
0.011529215	20	1734.7234759768	86.7361737988
0.009223372	20	2168.404344971	108.4202172486
0.0073786976	20	2710.5054312138	135.5252715607
0.0059029581	20	3388.1317890172	169.4065894509
0.0047223665	20	4235.1647362715	211.7582368136
0.0037778932	20	5293.9559203394	264.697796017
0.0030223145	20	6617.4449004242	330.8722450212
0.0024178516	20	8271.8061255303	413.5903062765
0.0019342813	20	10339.7576569128	516.9878828456
0.001547425	20	12924.697071141	646.2348535571
0.00123794	20	16155.8713389263	807.7935669463
0.000990352	20	20194.8391736579	1009.7419586829
0.0007922816	20	25243.5489670723	1262.1774483536
0.0006338253	20	31554.4362088404	1577.721810442
0.0005070602	20	39443.0452610505	1972.1522630525
0.0004056482	20	49303.8065763131	2465.1903288157
0.0003245186	20	61629.7582203914	3081.4879110196
0.0002596148	20	77037.1977754893	3851.8598887745
0.0002076919	20	96296.4972193616	4814.8248609681
0.0001661535	20	120370.621524202	6018.5310762101
0.0001329228	20	150463.276905252	7523.1638452626
0.0001063382	20	188079.096131566	9403.9548065783
8.5070592E-005	20	235098.870164457	11754.9435082228
6.8056473E-005	20	293873.587705571	14693.6793852786

Table 5.5: Table showing the values being fed into dnsblast

*multiplier = multiplier * 0.8*

Total represents the total amount of queries to be made during the 20 second duration, duration remains constant throughout the experiment.

total = 20 / multiplier

QPS (Queries per second) represents the number of queries sent from the polling server to the DNS server per second.

QPS = total / 20

The experiment was conducted for 15 minutes and 20 seconds before it was terminated. It seemed (from observing the output from 'top' and 'vmstat' and the DNS server host), that a maximum threshold is reached where raising the amount of requests per second on an already loaded server seemed to make no difference. This was probably an issue relating to the network I/O of the Alix system board as they do not possess separate network and I/O controllers and everything is managed by the CPU.

It should also be noted here that these tests were not being conducted over a wireless connection. The results received when spamming the DNS server over a wireless connection were far off from even the first attempts before we started using DNS blast. This is probably due to greater latency when connecting via wireless.

This is confirmed, when we examine the average result returned from ping tests (17ms) to various nodes in the network. As this network latency would be a predefined value in determining our maximum rate at which we could expect results to be sent and received. As this was more a test of the maximum load on DNS server and not a test involving the DNS server running in the wild, the fact that we tested this over a cabled connection is irrelevant.

We can see by examining figure 5.4 that the response from the DNS server becomes very unstable as the number of requests per second increases. We can cross examine this with the results being made visible in figure 5.5 and we can see that the CPU idle drops to zero eventually and does recover during the tests.

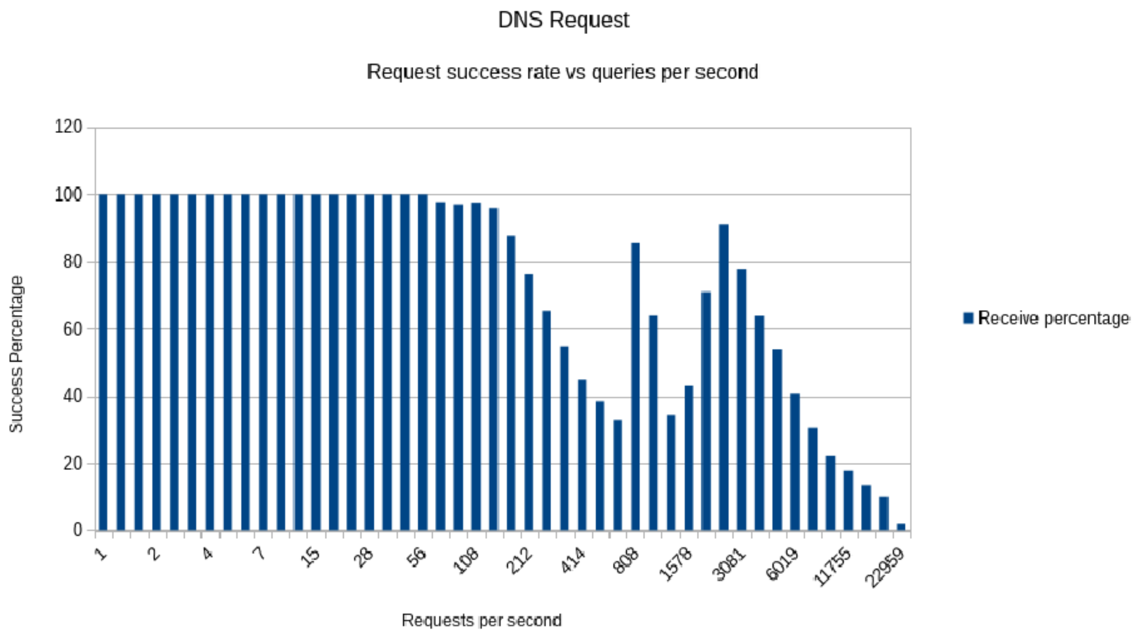


Figure 5.4: Graph showing the DNS query success rate versus the number of requests per second sent by the polling server

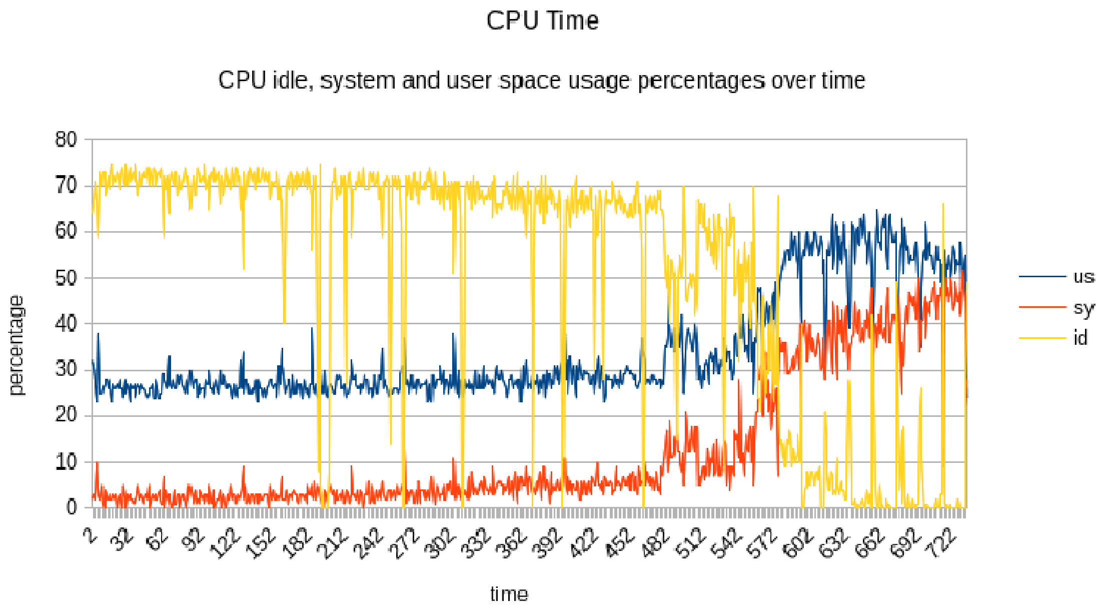


Figure 5.5: CPU utilisation of the DNS server over time during the tests

It does fine initially but as it nears 482 seconds (136 169 requests per second) the system starts to keel over and the reliability falls. The system seems to recover somewhat at around 808 requests per second. There is no way to reliably determine the cause of this. There seems to be a trend downwards but the spikes only confirm the assertion that the system becomes unstable/unreliable as the number of requests increase.

The system therefore seems to only be able to reliably respond to requests that are coming in every 5 - 7 milliseconds. Everything else after that seems to indicate a sharp drop in reliability of the system. Table 5.6 is included to show the related monitor levels to the corresponding query per second value which was fed into 'dnsblast'.

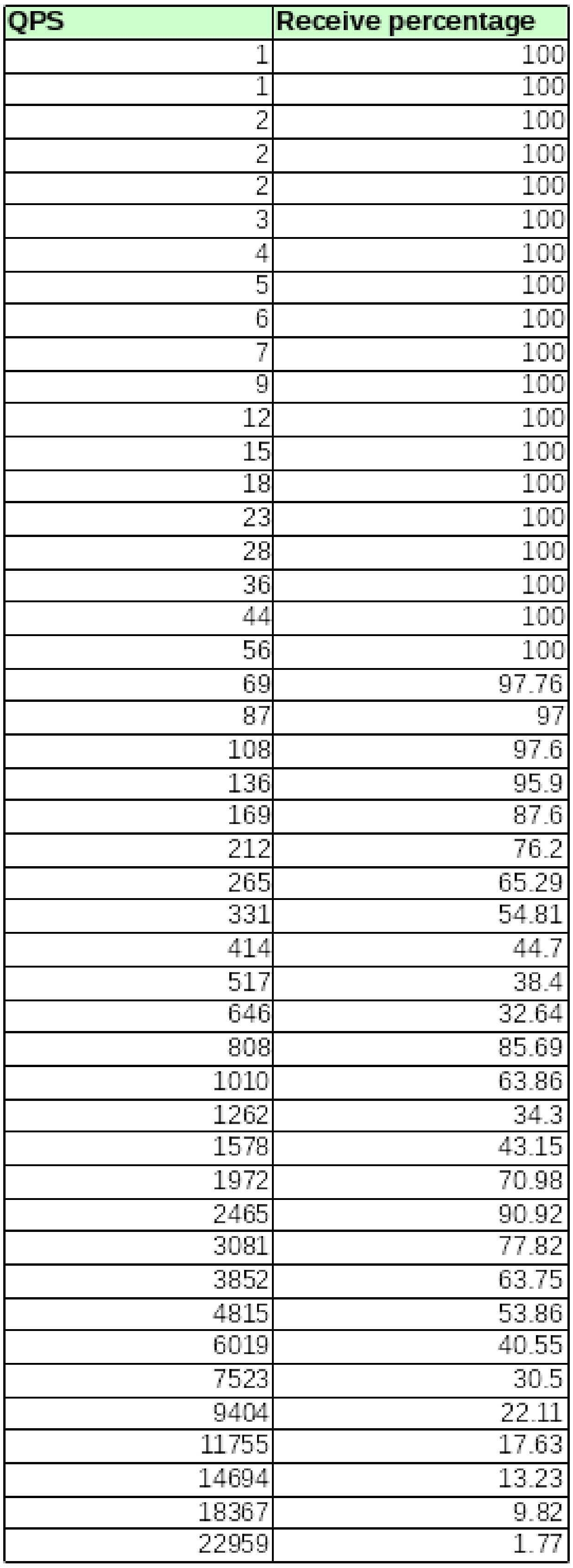
5.3.2 Performance tests

The performance tests were conducted twice. We had initially loaded the the desktop machines with a much higher 'penalty_max' value. This meant that these host machines could take a lot more punishment by way of more processes being allocated to it. The Alix boards also had the 'handicap' of by default having higher penalty values by not having any swap space on the disks. This meant that by default the swap value in our penalty calculations would be set to the value of 1.

This was expected behaviour of the system. The overzealous 'penalty_max' values did attribute those systems being loaded with much more processes than what they could handle. And it was noticed that the machines were thrashing badly. This was partly due to the misconfiguration of the stress utility. We had loaded much higher performance penalty values that what should have been.

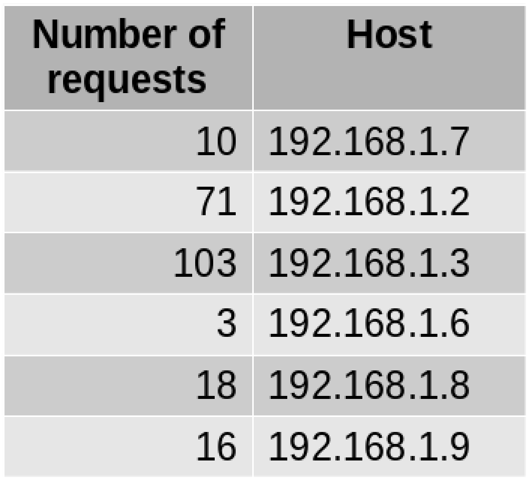
We also loaded them disproportionately. Some tasks were set out to max out CPU for 90 seconds and make no subsequent impact on memory and swap. This caused the systems to be loaded with more processes than what they were able to handle. So it seems that this was a combination of bad 'penalty_max' values and poor configuration of the 'stress-ng' application.

Table 5.7 displays the number of queries made to each host in the network. As we can see, high volumes of calls are made to the two desktop machines when compared



QPS	Receive percentage
1	100
1	100
2	100
2	100
2	100
3	100
4	100
5	100
6	100
7	100
9	100
12	100
15	100
18	100
23	100
28	100
36	100
44	100
56	100
69	97.76
87	97
108	97.6
136	95.9
169	87.6
212	76.2
265	65.29
331	54.81
414	44.7
517	38.4
646	32.64
808	85.69
1010	63.86
1262	34.3
1578	43.15
1972	70.98
2465	90.92
3081	77.82
3852	63.75
4815	53.86
6019	40.55
7523	30.5
9404	22.11
11755	17.63
14694	13.23
18367	9.82
22959	1.77

Figure 5.6: Queries per second and their corresponding received response percentage levels



Number of requests	Host
10	192.168.1.7
71	192.168.1.2
103	192.168.1.3
3	192.168.1.6
18	192.168.1.8
16	192.168.1.9

Figure 5.7: Table displaying the number of service requests made to each host in the test

to the Alix system boards. Table 5.8 also gives us insight into the behaviour of the DNS server. We can see that even though the systems do get loaded and absorbing all the service requests. Eventually the 'penalty_max' values for each host are reached and the requests get denied to the querying service; in this case our bash/python script.

Of the overall request sent, we can see that the majority of the requests got denied by the polling server. We can confirm the performance hits taken by the various machines in the network by comparing the various CPU charts in included.

As we can see from the graphs, the two desktop machines are taking a lot of punishment. While the relative 'penalty_max' values protect the Alix system boards from experiencing the same sort of punishment as the desktops. The two desktop machines were extremely unresponsive and it was quite difficult to stop the tests towards the end of the experiment as they were too busy trying to complete the 'stress-ng' tasks that had been allocated to them.

The experiment was conducted again, this time the desktop machines had their penalty values adjust to be the same as that of the Alix system boards (60). The machines were all rebooted to clear up any resources that might still be occupied before the experiments were conducted again.

When we compare the service requests in the network we can see that the load

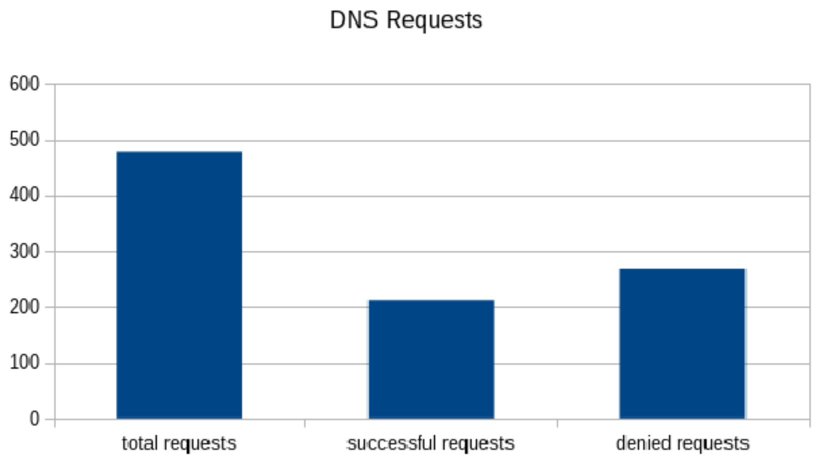
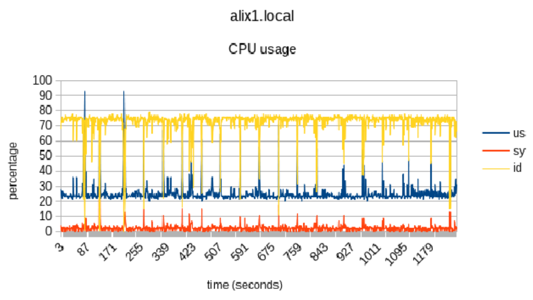
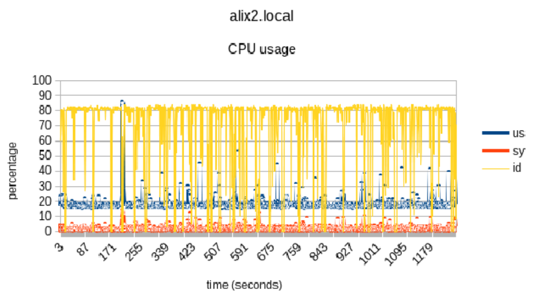


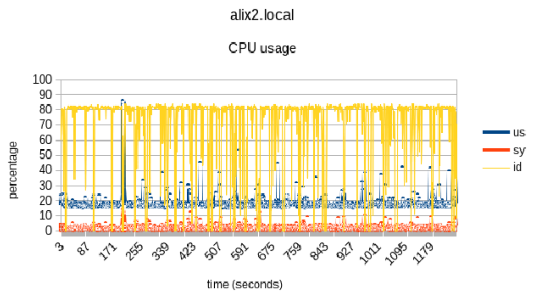
Figure 5.8: Graph displaying requests made during the experiment



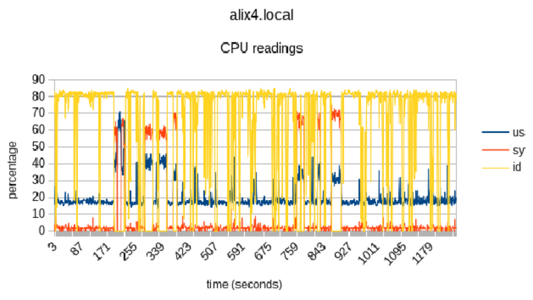
(a) CPU usage on alix1.local



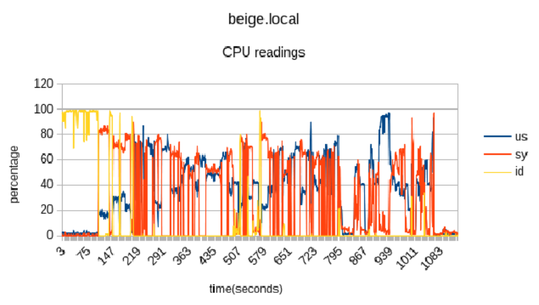
(b) CPU usage on alix2.local



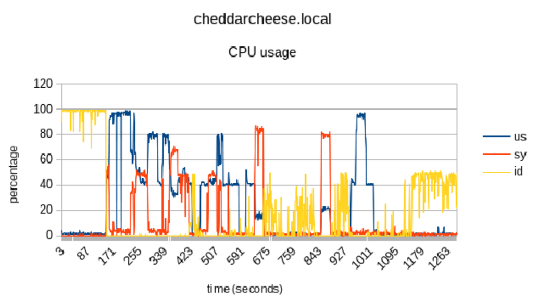
(c) CPU usage on alix3.local



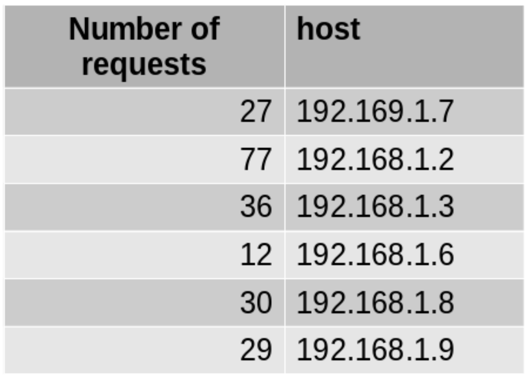
(d) CPU usage on alix4.local



(e) CPU usage on beige.local



(f) CPU usage on cheddarcheese.local



Number of requests	host
27	192.169.1.7
77	192.168.1.2
36	192.168.1.3
12	192.168.1.6
30	192.168.1.8
29	192.168.1.9

Figure 5.9: The number of service requests made to each host

now appears to be much more distributed than before. The desktop machines still absorb considerably more tasks than the Alix system boards. It is assumed that due the the desktop machines having a much lower penalty than the Alix system boards would attribute it to being allocated more tasks. Also, having a much lower ‘penalty_max’ value means that the device wont be allocated too many heavy tasks.

This allows it to both complete tasks with relative ease and free itself up for other tasks. The Alix system boards in all likelihood were never even hit with a request to load one of the more weighty tasks (map, calendar, store).

We can see from figure 5.10 that the system seems to perform relatively similarly. We can examine this further to note the actual difference between the number of calls made by looking at the tables 5.6a and 5.6b. The first run allows 10 more requests than the 2nd run of tests which is 2% more. It is not sure whether this value is statistically significant as the experiment would need to be run a deal many more times in order to determine this. But it is assumed that since the polls were sent out at random, the distribution should be fairly uniform.

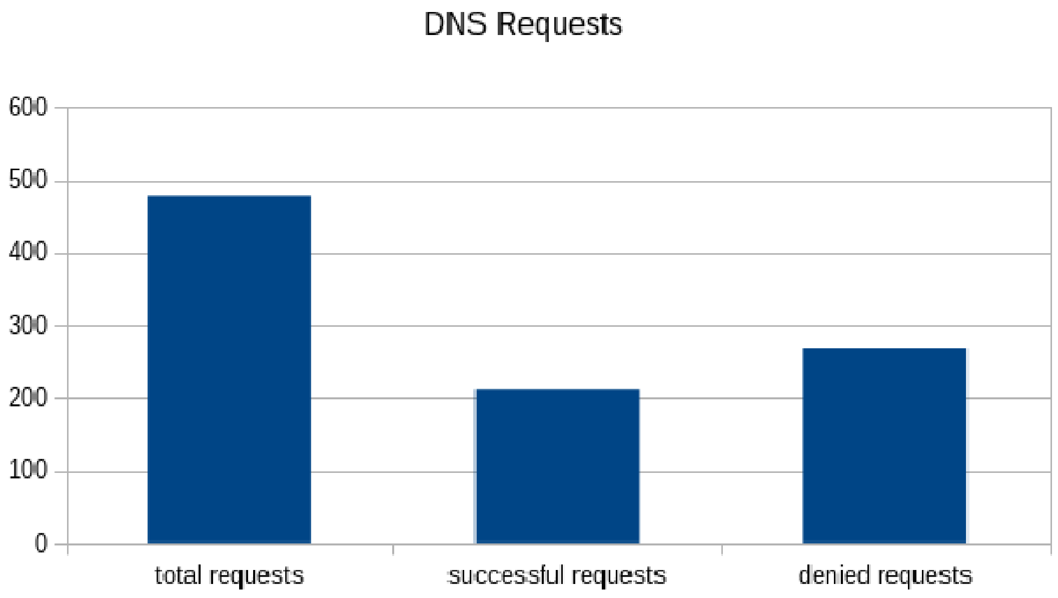
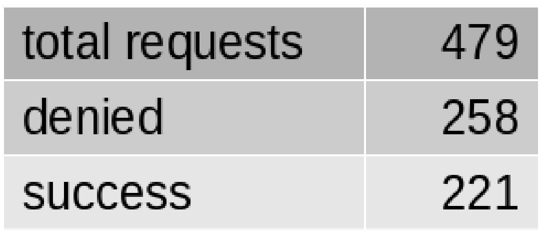
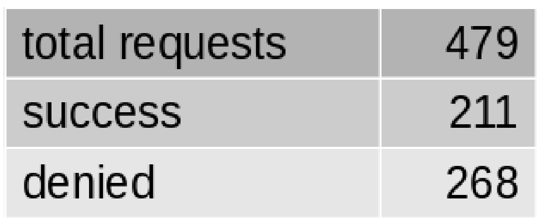


Figure 5.10: Graph displaying the number of requests made during the second run of the experiment



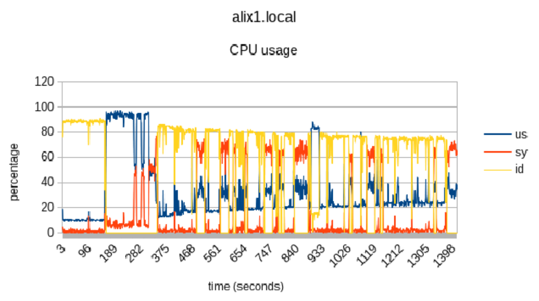
total requests	479
denied	258
success	221

(a) Service request 1st run

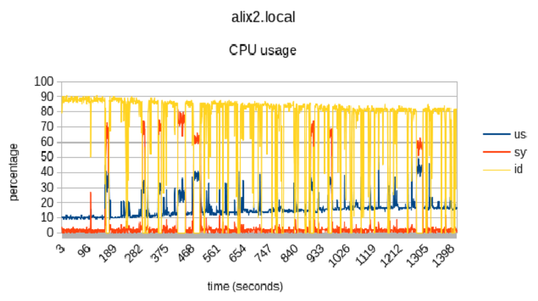


total requests	479
success	211
denied	268

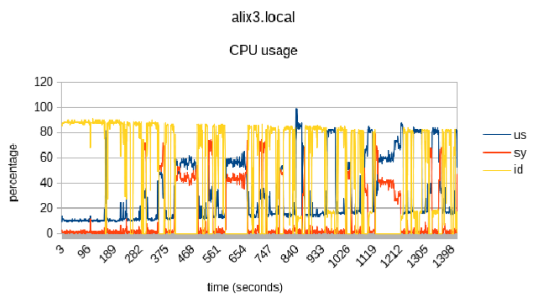
(b) Service request 2nd run



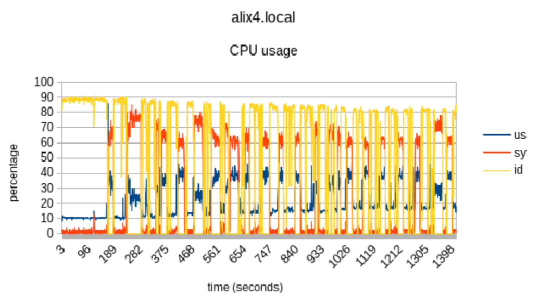
(a) CPU usage on alix1.local



(b) CPU usage on alix2.local

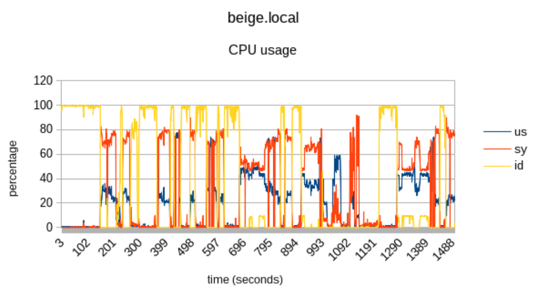


(c) CPU usage on alix3.local

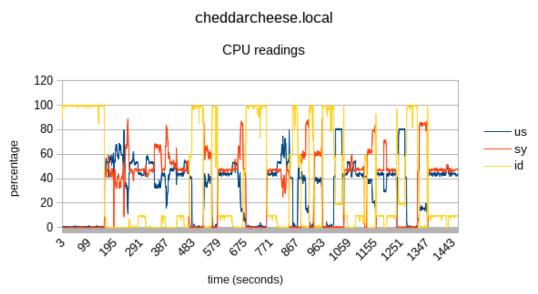


(d) CPU usage on alix4.local

The performance of the various nodes in the network can once again be examined by the performance graphs that follow. We can see that the Alix system boards perform relatively the same when to the previous run. The desktop machines though, seem to have a considerable amount of more free resources available to them. This confirms the view that fewer higher weight tasks were allocated during this run of the experiment.



(e) CPU usage on beige.local



(f) CPU usage on cheddarcheese.local

5.4 Conclusion

Ray Bellis, on his implementation of the 'evldns' solution that he created was able to achieve a query to answer rate of 60 000 queries per second. Ray however, was testing on an HP DL385 server. Somewhat considerably faster than our Alix system board. Also, it is not entirely apparent how Ray tested this but we will consider that a similar technique to ours was applied.

Our solution was only able to reach 136 queries (95% accuracy) per second before the great decline in reliability started kicking in. Considerably speaking though, the system performs reasonably well.

Especially since will most likely not be able to receive data even close to the max value(136-169 DNS queries per second) we tested in a wireless networked environment. The various controllers on the device are simply far too slow to keep up with the amount of network traffic coming and the packets just get dropped or lost.

Even if the solution was employed in a wired environment, the application of this solution would require deployment in a highly specialised network environment, so it is unlikely that this amount would ever be reached in a wired environment either unless the network falls victim to various network attacks.

That however would become the subject of a different investigation and beyond the scope of this project. This brings on questions of lightweight security practices, which is not handled in this text either. It is duly noted though that these investigations must be made in order to complete the solution.

The performance tests indicate the systems ability to distribute the load among various nodes in a wirelessly networked environment.

We had initially set too high of a 'penalty_max' value for the desktop machines in the network. This had side effect is that the two host seem to absorb most of the system calls made in the network and having the effect of not evenly distributing the tasks. This does however give us a key insight into the solutions behaviour in a heterogeneous network where multiple devices are connected.

Also having high 'penalty_max' values should really be discouraged unless the machine is really high performance. This is because the current penalty calculation is done with a relative comparison of performance to the 'penalty_max' value. However, the services, when being allocated to hosts in the network are adding a fixed not a scaled penalty value. Which is the correct intended behaviour. But this means that when the 'penalty_max' value is decided upon it must be done in a way which most accurately describes the performance state of the machine, especially in relation to the processes it must run.

This however should be a secondary concern. Ideally, we would like the solution to automatically calculate the impact on the system the allocated process occupies and dynamically manage the penalty values according to each host. This will prevent any issues with misconfigured hosts from being a problem in the network as we experienced. This too will be the subject of a future investigation and optimisations in the inner workings of LRAP.

We noticed the issue of calculating the system availability. Using the CPU load in itself is not enough and it seems here we reach the limits of our IDLE calculations, where on the desktop machine, because of a misconfigured 'penalty_max' value; systems were getting way more requests than what they could handle.

This should have had the effect of preventing additional processes from spawning due to increasing the system resources, but the 'stress-ng' process was not configured to add much by way of RAM and SWAP ballooning. Meaning the system still had a relatively good score when compared to the Alix system boards, which already had the penalty handicap of appearing to have its SWAP space full. Not to mention the operating system is loaded in half of the systems memory leaving around half to a quarter for the system to play with.

Its still not convincing to say whether this case is alluding to the penalty calculation needed some more rework, or if the way performance is monitored with the LNMP process needs to be re-examined. This too should be the subject of future investigations into optimising this solution.

Chapter 6

Conclusion

This thesis has contributed toward a lightweight storage and service infrastructure for cloud computing that is built around BigCouch and a network management protocol that uses two protocols: The LNMP for cloud network monitoring and the LRAP for grid network service allocation.

The initial results showing off the performance of the proposed protocols, LNMP and LRAP, are very promising. Both protocols show promise and their combination in running in tandem with each other allow for easy service provisioning on the network.

Each section of the project has displayed its own promise in its implementation. Chapter 5, while showing the ability of the protocol to effectively disseminate tasks to the various worker nodes in the network, it also presents an effective display of all the parts in the system working in concert with each other.

Both the distributed nature of the databases being used and the monitoring service which was feeding the allocation agent the updated performance values, were not necessary. The protocol operates independently of these conditions being true.

The service should be expanded further though. At the moment no network agent discovery is present. Currently it does feed into Avahi, which proves to be an excellent tool, but does not take it any further than this. Network discovery would be a great added feature, because once this is present the cloud can then start performing more interesting functionality, such as the auto assimilation of networks.

Through auto discovery nodes could find each other and automatically construct the grid.

Being able to move the grid coordinator to a different node automatically depending on the network load or even network topology would also be possible. The possibility of networks dynamically growing and then perhaps even segmenting itself to contain multiple coordinators would also be an interesting direction to take.

These little clouds of computing resources all operating together on a sea of wired and wireless connections, working in concert to perform what can be only the most imaginative step forward in our current age of digital watch dogs and growing privacy concerns.

Data requires liberation from the costly data centres where machine learning techniques are put to work mining the meta data of our lives. The only way we get to transcend our current mode of operation is to develop the means to escape these shackles.

The project has thus framed itself in a very acute context but the ideas explored herein should be expanded ever further. We hope to have illuminated the reader in these concluding remarks to a more subtle vision which has been the underlying aspiration of the project.

The undertaking of a work such as this is quite interesting in and of itself. One hopes that when starting the journey into research, that you start asking questions to yourself and not just about the work. "What is the point of this work? Why did I start? And after 15 months of working on this, why am I still here at the end? I am no longer the same person that started out.What would motivate me to continue with this vision?" These are questions which I have had to answer and relate to in my work. The embodiment of this work is hopefully reflected herein as well as any experienced personal growth associated with it.

6.1 Reflection on the work

We note possible shortcomings on the ideas proposed and note the over ambitious nature which this projects lends itself to be. Either that or the author reveals a lack of imagination in conceiving possible greatness. The protocol described in here should contain much more detail and being a protocol would require a somewhat different format of proposal. What we have here is an draft outline of what should be a more complete description. But the ideas discussed should illuminate readers into the framework being developed here. The use of the word protocol in this sense would be somewhat incorrect and instead a framework is proposed.

Separately it should also be noted that the project's predominantly focus relates strongly to issues playing in the low power wireless space. Desktop machines are however added wherever possible to introduce heterogeneity into the solution space as well. The wish was to not constrain the solution developed here to the resource constrained environment, but wished to act as a solution that could be applied on a much broader scale.

This thesis also freely uses the words grid, community cloud and participatory cloud interchangeably. These all ostensibly infer similar meaning throughout the document.

References

[1] W. A. Faunce, "Automation and the division of labour, social problems," August 1965.

[2] L. Burke, "Report says e-voting is unsafe," April 2000. Retrieved 21 August 2014, http://archive.wired.com/politics/law/news/2000/07/37504.

[3] J. A. et al, "A remote surgery experiment between japan and thailand over internet using a low latency codec system," 2007.

[4] P. T. Park, "Case study: Building your own data center vs. buying colocation services." Retrieved 21 August 2014, http://www.vnetusa.com/uploads/PTP_-_build_vs_buy.pdf.

[5] R. B. et Al., "Cloudbus Toolkit for Market-Oriented Cloud Computing," in *CloudCom*, p. 2444, 2009.

[6] R. B. et Al, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *HPCC 08. 10th IEEE International Conference on High Performance Computing*, 2008.

[7] National Institute of Standards and Technology, "NIST definition of Cloud Computing," September 2011. Retrieved 19 August 2014, http://csrc.nist.gov/publications/nistpubs/800- 145/SP800-145.pdf.

[8] Gerard Briscoe and Alexandros Marino, "Community Cloud Computing," in *First International Conference on Cloud Computing*, 2009.

[9] R. B. et Al., "InterCloud: Utility-Oriented Federation of Cloud Computing

Environments for Scaling of Application Services," in *ICA3PP 2010, Part I*, p. 13 31, 2010.

[10] T. H. et Al, "A Hybrid Articial Intelligence System for Assistance in Remote Monitoring of Heart Patients," in *HAIS 2011, Part II*, p. 413 420, 2011.

[11] M. Zennaro, B. Pehrson, and A. Bagula, "wireless sensor networks: a great opportunity for researchers in developing countries".," in *In the Proceedings of WCITD2008 Conference, Pretoria, South Africa*, 2008.

[12] A. Bagula, M. Zennaro, G. Inggs, S. Scott, and D. Gascon, "ubiquitous sensor networking for development (usn4d): An application to pollution monitoring,," in *Sensors ISSN 1424-8220, Vol 12, Pages 391-414; doi:10.3390/s12010039.*

[13] M. Mafuta, M. Zennaro, A. B. Bagula, G. W. Ault, H. S. H. Gombachika, and T. Chadza, "Successful deployment of a wireless sensor network for precision agriculture in malawi," in *International Journal of Distributed Sensor Networks 04/2013*, 2013.

[14] M. Zennaro, G. D. A. Floros, T. Sun, Z. Cao, C. Huang, M. Bahader, H. Ntareme, and A. Bagula, "on the design of a water quality wireless sensor network (wqwsn): an application to water quality monitoring in malawi," in *in the proceedings of the IEEE NGWMN 2009 conference*, 2009.

[15] M. Masinde and A. Bagula, "itiki: bridge between african indigenous knowledge and modern science of drought prediction,," in *Knowledge Management for Development Journal 7 (3), 274-290.*

[16] C. N. et Al., "Secure Cloud-based Medical Data Exchange," in *erschienen im Tagungsband der INFORMATIK*, p. 192, 2011.

[17] T. Mullins, J. Martin, and A. Bagula, "Participatory Cloud Computing : A Smart Middleware for the Internet-of-Things," *SAICSIT*, 2013.

[18] T. Mullins and B. Bagula, "Monitoring Community Clouds: The Lightweight Network Management Protocol.," *IEEE-UIC*, 2013.

[19] C. S. N. Kushalnagar, G. Montenegro, "RFC 4919," 2007. Retrieved 21 August 2014, http://datatracker.ietf.org/doc/rfc4919/.

[20] L. A. N. Man, S. Committee, and I. Computer, *Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) S pecifications*, vol. 2012. 2012.

[21] R. Wolski, N. Spring, and J. Hayes, "Predicting the CPU Availability of Time-shared Unix Systems on the Computational Grid *," 2014.

[22] D. Stenberg., "curl and libcurl," 2014. Retrieved 21 August 2014, http://curl.haxx.se/.

[23] N. I. o. S. Department and Technology, "SNMP host mibs," 2013. Retrieved 19 August 2014, http://net-snmp.sourceforge.net/docs/mibs/host.html.

[24] M. R. McCloghrie and K., "Structure and identification of management information for tcp/ip-based internets," 1988.

[25] G. N. S. Srivastava, "Enhancing the efficiency of secure network monitoring through mobile agents," in *First International Conference, CloudCom*, pp. 141–148, 2010.

[26] Hyperic, "SIGAR," 2014. Retrieved 19 August 2014, http://www.hyperic.com/products/sigar.

[27] L. Carlos, D. Junges, and I. L. Martins, "Achieving High Availability , Scalable Storage and Performance at Portal do Aluno - Distributed Databases Overview Study -," 2009.

[28] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo : Amazon s Highly Available Key-value Store," pp. 205–220, 2007.

[29] CloudAnt, "NOSQL Databases," 2014. Retrieved 19 August 2014, http://nosql-database.org/.

[30] CloudAnt, “BigCouch: API,” 2012. Retrieved 19 August 2014, http://bigcouch.cloudant.com/api.

[31] solid IT, “DB-Engines Ranking - popularity ranking of database management systems,” 2014. Retrived 19 August 2014, http://db-engines.com/en/ranking.

[32] Transcend, “JetFlash600-USB Flash Drives.,” 2013. Retrieved 19 August 2014, http://www.transcend-info.com/Products/No-267.

[33] Oracle, “MySQL :: MySQL 5.1 Reference Manual :: 8.6.1 The MyISAM Key Cache,” 2014.

[34] Apache, “3.3. CouchDB HTTP Server Apache CouchDB 1.7.0 Documentation,” 2014. Retrieved 19 August 2014, http://couchdb.readthedocs.org/en/latest/config/http.html.

[35] CloudAnt, “couch_httpd.erl,” 2012. Retrieved 19 August 2014, https://github.com/cloudant/bigcouch/blob/master/apps/couch/src/couch_httpd.erl.

[36] Ericsson, “Erlang - Processes,” 2014. Retrieved 19 August 2014, http://www.erlang.org/doc/efficiency_guide/processes.html.

[37] J. Barr, “Amazon ec2 beta,” August 2006. Retrieved 21 August 2014, http://aws.ambzon.com/blogs/aws/amazon_ec2_beta/.

[38] Gregory P. Smith, “subprocess32 3.2.6 : Python Package Index,” 2014. Retrieved 19 August 2014, https://pypi.python.org/pypi/subprocess32/.

[39] A. Waterland, “stress,” 2014. Retrieved 19 August 2014, http://people.seas.harvard.edu/ apw/.

[40] Colin King, “stress-ng,” 2014. Retrieved 19 August 2014, http://kernel.ubuntu.com/git?p=cking/stress-ng.git;a=summary.

[41] F. Denis, “dnsblast,” 2014. Retrieved 19 August 2014, https://github.com/jedisct1/dnsblast.

Appendix A

Sample configuration file

```
#Sample configuration file Comments are denoted with the #
#storage size in gigs
storage_size=4

#either fqd or ip address, along with port to connect to
hostname=127.0.0.1
hostport=30000

#1 is server mode, 2 is client mode
type=2

#not required in this release
gui=false

#the port to listen on locally
port=29999

#the storage dir to store files in, using relative pathing
storage_dir=storage

#the n-copies variable for determining how many copies of files to
```

```
#store
n-copies=3

#a dynamically allocated uuid. auto generated and inserted if the
#file does not contain a valid uuid
uuid=43412349a9b99accc89812

#used to detirmine if we are going to store the stats that we
#collect or send them to the server
store_stats=true

#If we are going to store the stats, are we going to do so in a
#local only db. If that is the case we are going to have to notify
#the server when alerts get raised.
local_db=true

#If we require a special db access string, it must be defined here.
database_type_string=bigcouch
database_host=localhost

#If we want the DNS server to run, set the following to true;
rundns=true;

#The following operator sets the search domain the server runs in.
#Any requests not in the search domain, automatically get rejected.
#For more information, check the wiki.
searchdomain=site

#the maximum penalty value allowed on the host
penalty_max=60
```

Appendix B

Sample service declaration file

```
<xml>
    <service>
        <name>asterisk</name>
        <weight>3</weight>
        <port>5060</port>
        <service-url></service-url>
    </service>
    <service>
        <name>storage</name>
        <weight>2</weight>
        <port></port>
        <service-url></service-url>
    </service>
    .
    .
    .
</xml>
```

Appendix C

Sample BigCouch View

```
{
   "_id": "_design/systemreadings",
   "_rev": "1-d13a953963f47e3412f20a4d9b5541e6",
   "language": "JavaScript",
   "views": {
       "allreadings": {
           "map": "function(doc){if(doc.type ==’reading’){emit(date,doc);}}"
       },
       "everything": {
           "map": "function(doc){emit(null,doc);}"
       },
       "latest": {
           "map": "function(doc){emit(doc.uuid,doc);}"
       }
   }
}
```

Appendix D

Code Samples

D.1 Convert partition

```
fsck -n /dev/sdb1

tune2fs -O ^has_journal /dev/sdb1

e2fsck -f /dev/sdb1

resize2fs /dev/sdb1 4000M

fdisk /dev/sdb

fsck -n /dev/sdb1

tune2fs -j /dev/sdb1
```

D.2 Database scripts

BigCouch insert

```
for i in {1..10000}

do

        echotext={\"var1\":\"$i\",\"var2\":\"$i\",\"var3\":\"$i\",\"var4\":\"$i\

        curl -X PUT http://localhost:5986/test_db/doc_$i -H content-type:applica

done
```

BigCouch Read

```
for i in {1..10000}

do

        curl http://localhost:5984/test_db/doc_$i

done
```

MySQL insert

```
for i in {1..10000}

do

        mysql -u root -e "insert into test_db.testdata (val1,val2,val3,val4,val5

done
```

MySQL read

```
for i in {1..10000}

do

        mysql -u root -e "select * from test_db.testdata where id=$i"

done
```

Appendix E

Service Diagrams

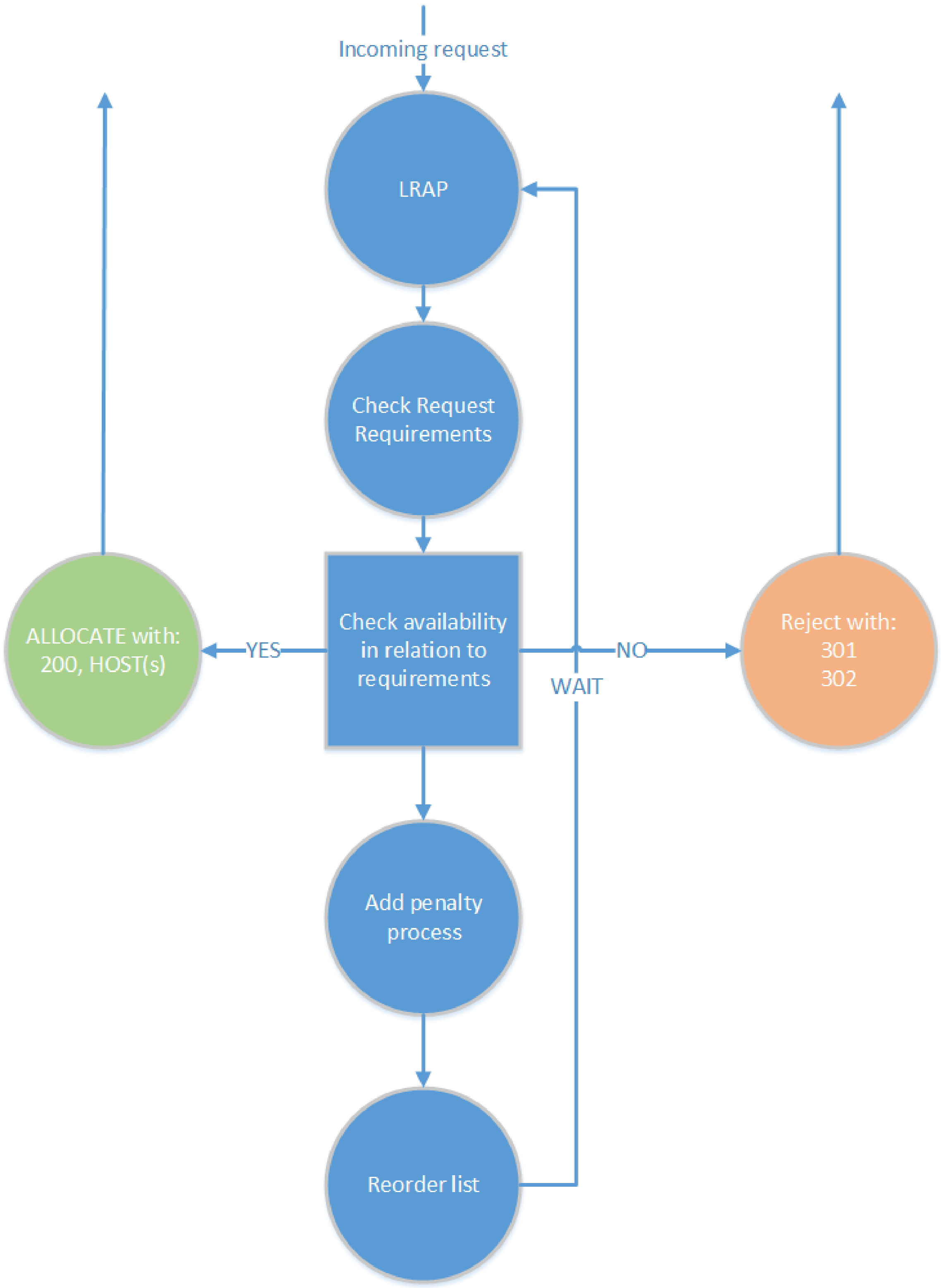

Incoming request
LRAP
Check Request Requirements
Check availability in relation to requirements
ALLOCATE with: 200, HOST(s)
YES
NO
WAIT
Reject with: 301 302
Add penalty process
Reorder list

Figure E.1: Allocation cycle

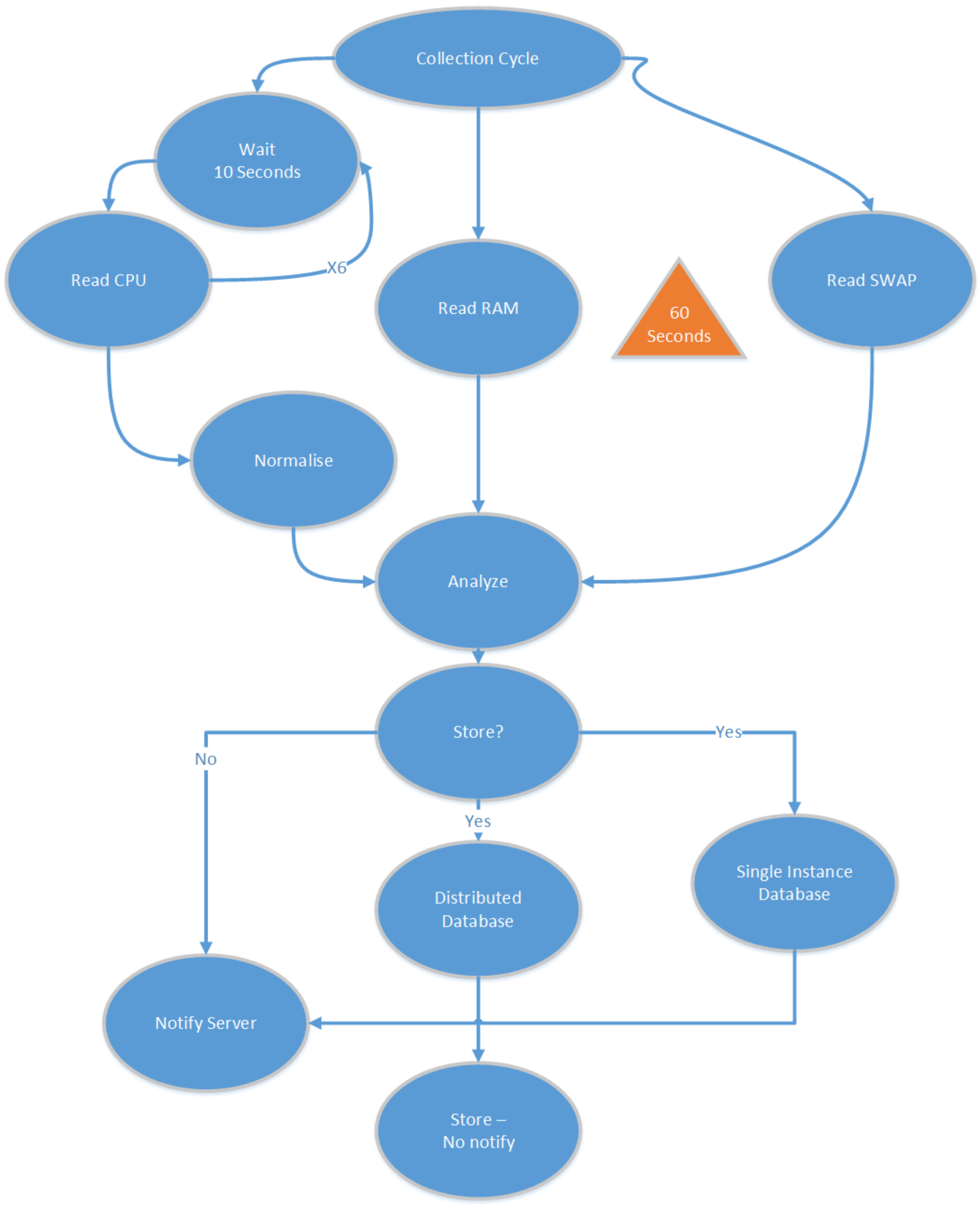


Figure E.2: Analyze cycle

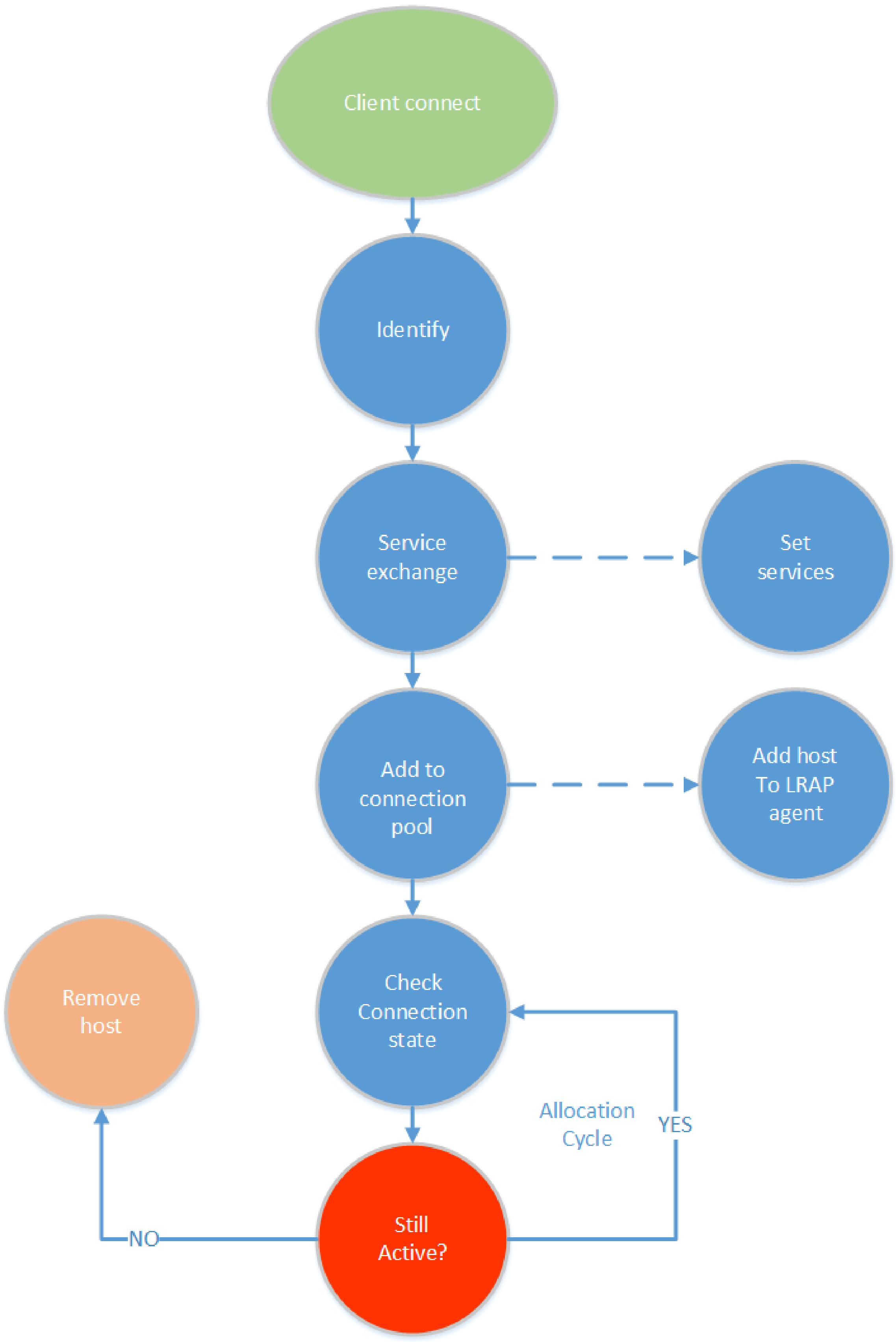


Figure E.3: Connection Flow