Towards a Lightweight Mobile Cloud

Dissertation Submitted in Part Fulfilment of the Requirements for the degree of Master in Computer Science

University of Dublin

Trinity College

2011

Paul Woods

Dissertation Supervisor: Dr. Siobhán Clarke

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

I also agree that the Library of the University of Dublin, Trinity College may lend or copy this dissertation upon request.

Signature:

_____

Paul Woods

Date: 29th August 2011

Acknowledgements

I would like to thank my supervisor, Dr. Siobhán Clarke, for her guidance and support throughout this dissertation. This support extended to weekly meetings in support of the exploratory analysis, design, implementation and evaluation.

In addition I'd like to acknowledge all of the lecturers I studied under for the M.Sc. in Networks & Distributed Systems and the six other students from the class. I am also grateful to several of the PhD students who took the time to discuss areas of interest. In this regard I especially thank Brendan Cody Kenny and Michael Clear.

Lastly I'd like to pay tribute to all of the great computer scientists, both past and present who inspired my participation on the program and who have contributed more than I could ever aspire to. I believe, and hope, that computer science, coupled with the harnessing of other disciplines, will provide a platform for modeling, managing and resolving many of the key problems we face in a structured and efficient manner.

Dissertation abstract:

Towards a lightweight mobile cloud

Author: Paul Woods

ID No. 10263756 1 RS

Submitted in part satisfaction of the
Masters Degree in Computer Science
Trinity College Dublin
August 2011

Abstract

A typical cloud-computing environment is characterised by the networking of tens of thousands of servers. Once deployed, such datacenters experience a relatively stable hardware and networking environment. Smartphones offer considerable processing, memory, storage and sensing resources, and are handheld, pocket-sized devices that users generally carry with them at all times. The vision for cloud computing could foster more flexible models, whereby general users, including mobile ones, could participate in the cloud as both providers and consumers of resources. If the cloud is to computing what the Internet is for data, the opportunity for mobile computers to participate in service provision should be investigated.

The motivation for this dissertation is threefold. First, it explores the forces that resulted in the emergence of cloud computing. Second, the implications of mobile server nodes within a cloud environment are considered. Finally, the dissertation focuses on implementing a mobile storage cloud. Such a system could support applications including disseminating critical information in a disaster zone, mobile distributed social networking, and sensor applications such as collecting data on traffic movements.

The background analysis indicates several developments that fostered the emergence of cloud computing. During the 1960s virtualisation was developed for allocating mainframe resources, but later waned due to the commoditisation of hardware, only to re-emerge for allocating computing resources within datacenters. Improvements throughout the 1980s and 1990s reduced commodity hardware and networking costs, stimulating migration to horizontally scalable datacenters. Finally, automating resource provisioning via online portals enabled the flexibility and efficiency of federated computing to become widely accessible.

Evaluating the implications of mobile server nodes highlights numerous constraints for implementing a mobile cloud. Most importantly it becomes evident that the traditional cloud architecture will not suffice. A smartphone will not currently support a hypervisor or multiple virtual machines. Mobile devices are not designed to act as always-on servers, as they are battery powered, with limited computing resources, and experience low and intermittent network bandwidth. A mobile cloud will therefore only support applications that are designed for such an operating environment. Service availability takes on a different meaning within a mobile cloud as nodes regularly arrive and churn off the network.

The implementation involved designing and testing a mobile storage cloud, named Icarus. Each participating node incorporates client, directory and storage functionality. Using smart directory logic to dynamically replicate both content and directory metadata, depending on content importance, allows for the efficient utilisation of resources within the network. This ensures content survives node failure or node departure. Once a file is located it can be retrieved on a peer-to-peer basis.

The evaluation was based on simulating networks of varying size, and demonstrated that the proposed Icarus architecture survives node churn, while providing reasonable availability and high performance. Dynamically replicating and distributing files and related URL metadata separately across the network improved availability by minimising the probability that all replicas are unavailable or that requested content cannot be located. Multi-hop replication of directory metadata reduced the storage and transmission overhead, relative to replicating content, providing a sizeable improvement in network performance. An Icarus network can be tuned to provide deterministic availability and performance.

Dissertation supervisor: Dr. Siobhan Clarke
Trinity College Dublin

CONTENTS

Index of tables, graphs, diagrams and pictures

1. INTRODUCTION

1.1. Context

Much of the literature relating to mobile cloud computing involves mobile devices acting as thin clients and leveraging datacenters for back-end computation. In practice mobile cloud based applications entail the static partitioning of applications where the mobile device is responsible for executing customer-facing tasks such as the user interface while the datacenter hosts the server, middleware and database capabilities. Smartphones and tablet computers come with onboard processing, memory and solid-state storage resources equivalent to reasonable specification laptops of just a few years ago. Such improvements in mobile computing are set to continue, witnessed by the recent announcement by Qualcomm, to offer a 1.5 GHz dual core processor[1] for mobile devices and Apple's shift to using multicore processors in the iPhone. In contrast to traditional computing, this new breed of computer is encapsulated in a handheld, pocket sized device that users carry with them everywhere, and all of the time.

Typical cloud computing environments involve the deployment of large-scale server infrastructure in a manner proprietary to service providers such as Google, IBM, Amazon, Apple and Facebook. These service providers manage a global network of datacenters, with each networked to combine the computing power of tens of thousands of commodity servers. Impressive as these warehouse scale facilities are, such cloud computing platforms are at an early stage of development as datacenters are largely isolated computing platforms used for the provision of specific applications, platforms or bare infrastructure resources.

---

[1] See http://www.qualcomm.com/videos/snapdragon-dual-core-explained for an explanation of the new duel-core mobile processor.

The vision for cloud computing should be more flexible than these models, whereby general users, including mobile ones, can participate in the cloud as both consumers and providers of resources. If the cloud is to computing what the Internet is for data, the opportunity for mobile computers to participate in service provision should be investigated. Though the potential for a mobile cloud has not gone unnoticed, with organisations such as NASA exhibiting interest (Warner and Karman 2010), there has been limited in-depth exploration of the opportunities and challenges that arise.

There is an increasingly blurred definition of a mobile device given the proliferation of smartphones, Internet accessible tablet computers, electronic book readers and other such devices. A potentially significant opportunity exists to harness the power of these devices whether for concurrent processing, federated storage or for the provision of personalised mobile web applications hosted directly on the smartphone[2]. The move from smartphones simply leveraging applications as a thin client, to hosting services directly, could lead to a wide variety of new and innovative systems architectures and applications. Such designs could have implications for how information is collected and analysed on a daily basis in the world around us and provide alternative solutions to existing applications, in order to reduce privacy and security threats.

1.2. Motivation

The motivation for this dissertation is threefold. First, it explores the forces that resulted in the emergence of cloud computing. Second, the implications of mobile server nodes within a

[2] ComScore data showed that US smartphone penetration increased by 60% in the three months ending December 2010 (absolute number was 63.2 million smartphones) versus the same period for 2009. In Q4 2010, 100.9 million smartphones shipped worldwide versus 92.1 million PCs.

cloud environment are considered. Finally, the dissertation focuses on implementing a mobile storage cloud. Such a system could support applications including disseminating critical information in a disaster zone, mobile distributed social networking, and sensor applications such as collecting data on traffic movements. The system, named Icarus, is novel to the extent that mobile nodes serve as contributors to the cloud rather than merely acting as thin clients to leverage traditional cloud resources. The focus of Icarus is to provide distributed storage utilising mobile nodes, in a manner that fosters file availability irrespective of node churn within a wireless network. In a disaster zone, Icarus could be used for the dissemination and collaboration on critical information or to provide real-time information on potential obstacles to emergency routes. All of these applications are possible given the proliferation of sensors such as GPS, thermometer and accelerometer available on most smartphones today.

This research is important given the improvements in smartphone technologies coupled with the advancements in wireless networking that has resulted in a changing landscape whereby personal mobile computing resources could be federated to provide a distributed mobile cloud computing facility. The federation of 5,000 modern smartphones, each with 64Gb of storage and 1Ghz processor could provide 320 Terabytes of storage and 5 Terahertz processing capacity. Furthermore these computational resources are set to improve considerably.

1.3. Structure of dissertation

The next section of the dissertation analyses the background and implications of cloud computing. An analysis of options for enabling mobile nodes to participate in a cloud computing environment are then explored with reference to the literature. It becomes apparent that a variety of options exist for implementing a mobile cloud. A design is then outlined for Icarus and the implementation of this design clearly documented. An evaluation of Icarus,

using extensive network simulations is then completed. The dissertation concludes with the key lessons from the analysis and implementation.

2. BACKGROUND

2.1: Evolution of cloud computing

The term ‘cloud computing’ has been enshrouded in much marketing hype. The CEO of Oracle, Larry Ellison famously lashed out at cloud computing in September 2008 noting:

"The interesting thing about cloud computing is that we've redefined cloud computing to include everything that we already do. I can't think of anything that isn't cloud computing with all of these announcements. The computer industry is the only industry that is more fashion-driven than women's fashion. Maybe I'm an idiot, but I have no idea what anyone is talking about. What is it? It's complete gibberish. It's insane. When is this idiocy going to stop?”

If the CEO of Oracle couldn’t clearly decipher whether cloud computing is a new paradigm of computing then its clear that the term needs to be demystified. This section will seek to provide clarity by providing a clear definition and overview of what cloud computing is and how it came about. This will necessitate a brief history of its evolution and a discussion of the service delivery models that will be important in considering how mobile devices could participate within a cloud-computing environment.

A key benefit of cloud computing is the ability to seamlessly access remote and distributed applications in a transparent manner. An important goal of any distributed system is the ability to present itself as a single computer system when in use, a requirement referred to as transparency (Tanenbaum and Van Steen 2006). Cloud computing was originally conceptualised in the 1960s when John McCarthy noted, “computation may someday be organized as a public utility.” Many of the modern characteristics of cloud computing such as elastic provisioning and public, private and community forms were also explored in Douglas

Parkhill's book (Parkhill 1966) on utility computing. During the 1980s (Hagman 1986) and 90s (Clark 1992), research focused on the opportunities to leverage idle workstation resources. There was a realization that servers were commonly being operated at around one tenth of operational efficiency as core applications were being run on a dedicated server, often with redundancy provided via another dedicated server. Research started to suggest that supercomputing capabilities could be delivered through the use of parallel architectures (Kung, et al. 1991). Other researchers noted that a typical workstation was much less expensive than the compute node of most massively parallel supercomputers of the time (Li, et al. 1993).

In parallel research was progressing on how to improve the networking of workstations such that they would have high bandwidth and low-latency interconnects. This would ultimately help to close the speed gap that existed between supercomputer interconnects and interconnecting a network of workstations. Researchers on the NOW project (Network of Workstations) in Berkeley (Anderson, et al. 1992), noted that high speed networks were becoming faster than disks and that parallel programs could use more processors. They specifically noted that the existing high-end computing architecture at the time had no "near commodity" component and that the cost of deploying infrastructure was prohibitive. As a result researchers felt that a network of commodity workstations was worthy of study as a way of performing large-scale computation at lower cost. At the time though, there were few applications that could leverage parallel processing. The NOW project research concluded that the computing food chain beyond 2004 would consist largely of networks of commodity computers. Research into the use of hypervisor software to provide virtual machines added further credence to cloud computing by providing an efficient mechanism for allocating virtual machine resources to end users (P. Barham, K. Dragovic, et al. 2003). A hypervisor presents a virtual operating platform allowing multiple guest operating systems to share the same server hardware.

Throughout this period research was also ongoing into grid computing. While in some respects similar to cloud computing, a grid differs in a number of fundamental ways. With

grid computation, an individual user seeks to consume a large amount of federated computational resources (Berman, Fox and Hey 2003). A processing task is then allocated to a network of remote computers and processing is completed either in the background or during otherwise idle time with inputs and outputs transmitted across a broadband connection.

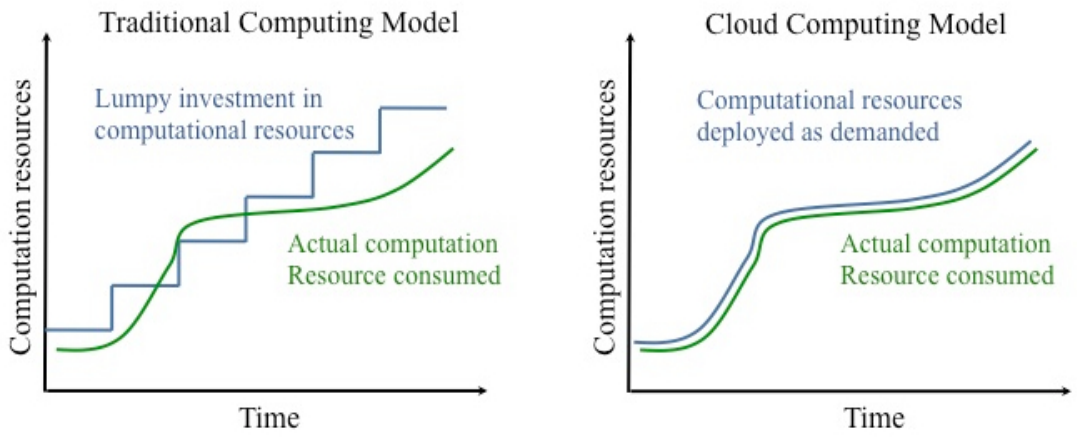
A virtual machine is the basic computational building block provided via cloud computing. Virtualization allows for server resources to be split and allocated into multiple logical servers. In essence virtualization abstracts the services running on a cloud from the underlying physical hardware. IBM developed technologies on virtualization in the 1960s and 1970s to allow multiple users share resources on a single machine (at the time for mainframe computers). Somewhat ironically, the progression towards affordable commodity computing resulted in a waning requirement for virtualization in the 1980s and 1990s as the requirement to share hardware decreased. More recently this has lead to the re-emergence of virtualisation in support of cloud computing as computing resources are amalgamated using warehouses of networked commodity servers. One of the beautiful benefits of virtualization is that when several virtual machines need to perform routine commands, they can be performed on the bare metal server rather than multiple times in the virtual machines, which would result in too high an overhead, adversely impacting performance. Prior to virtualization storage area networks (SANs) provided equivalent abstraction of storage by allowing the storage to be centralised for a large number of servers.

An issue that arises naturally from virtualization though is “virtual machine sprawl” (Sarathy, Purnednu and Miffilineni 2010). It is relatively simple to count the number of physical servers an organisation has, but more difficult to manage the total number of virtual machines used. Due to the widespread introduction of virtualization, there has been a significant increase in the demand for hardware and software support to optimise virtualisation. This has led to the introduction of logical routing across networks of virtual machines and a subsequent requirement for logical load balancers. As a result network traffic now needs to be managed and routed intra-server as well as inter-server. A consequence of this is that the cabling

previously required to interconnect servers has reduced given that multiple virtual machines are homed onto one physical server that uses a virtual internal network and that shares one physical Ethernet cable to connect to the broader rack of physical servers. Organisations outsourcing to cloud service providers may not therefore need to have the same networking expertise in-house. Server racks now come pre-cabled and on wheels, allowing them to be positioned and connected via the top-of-rack-switch to their respective datacenter cluster. This architecture results in the ability to build a highly stable physical hardware environment housing hundreds of thousands of servers that are interconnected via high-speed networking. A key operational metric within a cloud environment is the ability to efficiently run hardware across its asset life, which is usually three to four years. In this stable physical environment, only the virtualized instances change dynamically over time. The physical hardware and networking remains largely stable.

A key benefit of cloud computing is that the cost of running ten servers for one thousand hours can be the same as running one thousand servers for ten hours. This is a flexibility that had previously been unavailable to organizations. The motivation for the growth in cloud computing thus becomes clear given both the flexibility and operational efficiency that can be achieved relative to all organizations building their own separate IT infrastructure. In the diagram below the benefits accruing from cloud computing are illustrated. Traditionally organizational IT investment would have been characterized as lumpy given significant blocks of capacity are deployed to allow for headroom on future growth of the organisation, as illustrated in the left hand diagram. With cloud computing, given that very large scale networks of datacenters are deployed by third party organisations, an enterprise can now lease the computational capacity needed such that this capacity almost exactly fits the actual computational demands of the organisation.

Graph 1: Economics of cloud computing



Cloud computing enables organisations to utilize computational resources more efficiently and benefit from the greater economies of scale derived by allowing third party specialists to build cloud infrastructure that pools the computational requirements of many organizations. Thus infrastructure can be deployed at massive scale and separated from application development or deployment and leased on a variable pricing basis to produce economic efficiencies in the computational world somewhat analogous to what the industrial revolution brought to manufacturing. This can be witnessed with the move to very high scale datacenters and the rationalisation of sub-scale facilities[3]. Microsoft's datacenter in Ireland is 550,000 square foot. Meanwhile Google is estimated to be using 900,000 servers worldwide[4]. Adam Smith, the famous economist and pioneer of the division of labour would be pleased.

The history of cloud computing provides context for the modern definitions of cloud computing and service delivery models have largely been driven by the economics just discussed. A definition of cloud computing is provided by the U.S. National Institute of Standards and Technology (NIST 2011).

---

[3] The White House has recently announced a plan to shut down 373 datacenters by the end of 2012 with a longer term plan to close 800 by 2015. http://www.whitehouse.gov/the-press-office/2011/07/20/white-house-announces-plans-shut-down-hundreds-duplicative-data-centers-

[4] http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers/

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The NIST specification also outlines that cloud computing is characterised by the following:

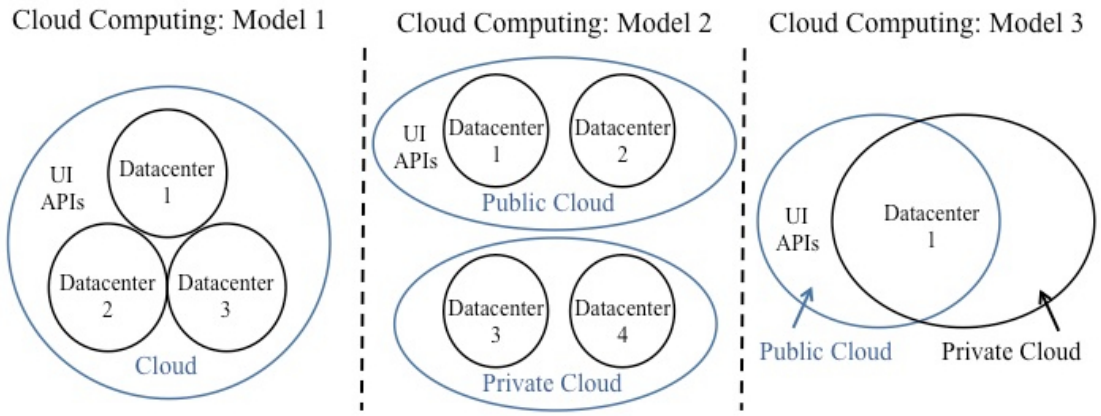
1. On-demand self-service. A user can sign up and consume services without long delays.
2. Broad network access. Ability to access the service via standard platforms (these could include desktop, laptop and mobile.).
3. Resources are pooled across multiple customers.
4. Rapid elasticity. Computational resources can be scaled in real-time to cope with demand peaks.
5. Consumption of services are measured. Billing is metered and delivered as a utility service.

The literature also suggests seven characteristics that make a large datacenter a cloud (Bernstein, et al. 2009). These are:

1. A pool of computing resources and services are employed that are shared amongst customers.
2. Services are charged for on an "as used" metered or capacity based pricing model.
3. Resources are distributed geographically but in a manner transparent to the customer (unless they request visibility).
4. Provisioning, configuration and deconfiguration are automated with no human assistance.
5. Resources are delivered virtually.
6. Physical infrastructure rarely changes. It is the virtually delivered resources that change constantly.
7. Resources can be of a physical or an abstract metaphor nature (such as message queue functions etc).

Both definitions of cloud computing share a number of similarities. Both Amazon Web Services (Amazon Web Services n.d.) and Google AppEngine (Google App Engine 2011) exhibit all five of the NIST and all seven of the Bernstein et al. characteristics. Reverting to Larry Ellison's quote, it is clear that an understanding of the difference between a datacenter and what is deemed to be cloud computing is required. The definitions indicate that a cloud could incorporate one or many datacenters that may be dispersed geographically to create a computing environment as represented by model 1 in the diagram below. In each case the cloud specific element represents the hypervisor software, suite of product offerings and self service front-end provided to the customers of the cloud provider.

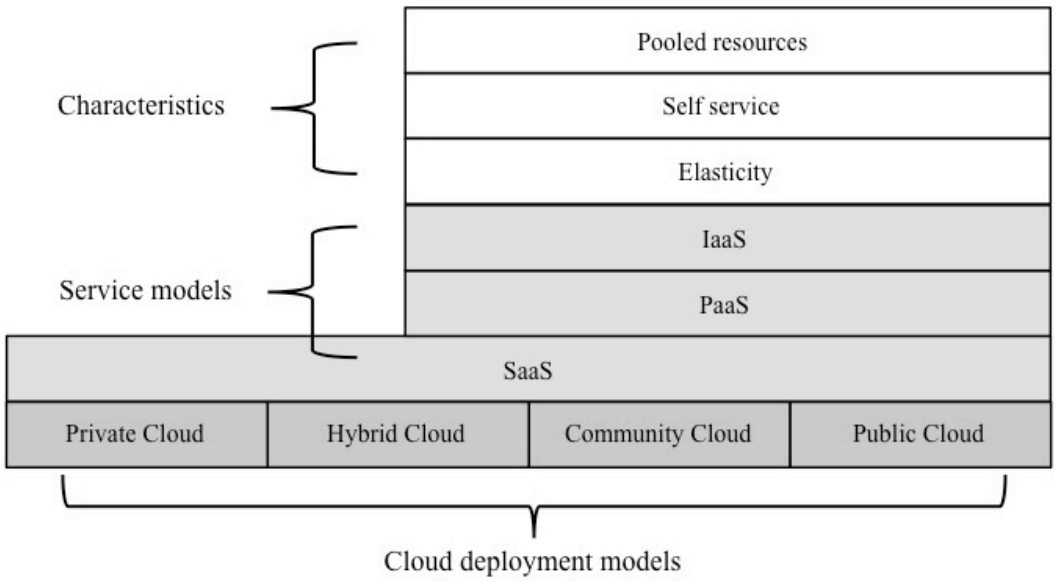
Diagram 1: Cloud Computing Models



The terms 'public cloud' and 'private cloud' have also emerged where a 'private cloud' refers to datacenter infrastructure that is privately used by an enterprise. Indeed such an enterprise may have dedicated datacenters (a private cloud) for its own internal operations and dedicated datacenters that are shared with other customers (a public cloud) as in the case of model 2. Amazon for example, required large scale datacenters initially to support its own online retail operations and later decided to leverage the infrastructure they had built by offering computing resources to other organizations. A provider of cloud computing resources with one or many datacenters might also partition some of the resources in its datacenter/s for other customers as in the case of model 3. The difference between a private and public datacenter at

the hardware level will largely be negligible. In certain instances though some physical changes could be made to tune the infrastructure either for the application being hosted or given the utilisation of the network. For example an application like MapReduce, used by Google, will utilise significant intradatacenter networking. In general however, the difference between a private and public cloud will relate to the scale of the infrastructure and the requirement to provide front-end tools and APIs to allow users to self-service the resources they lease from a public cloud, allowing for the efficient administration of virtual infrastructure. Thus far we have discussed public, private and hybrid clouds. Community clouds may also exist where entities with private computational resources such as universities can decide to federate resources into a computational cloud which can be used by all parties as needed. There are different models of service that can be provided from a user's perspective. The diagram below provides an overview of the deployment models, services models and the associated characteristics of cloud-computing.

Diagram 2: Cloud characteristics, service and deployment models



Software as a service (SaaS) is offered via cloud computing infrastructure were customers utilise applications hosted on the cloud. Typical examples of such applications include

Salesforce.com, Apple's iCloud and Google docs. SaaS applications are designed to support end-users, who access and consume the service over the Internet.

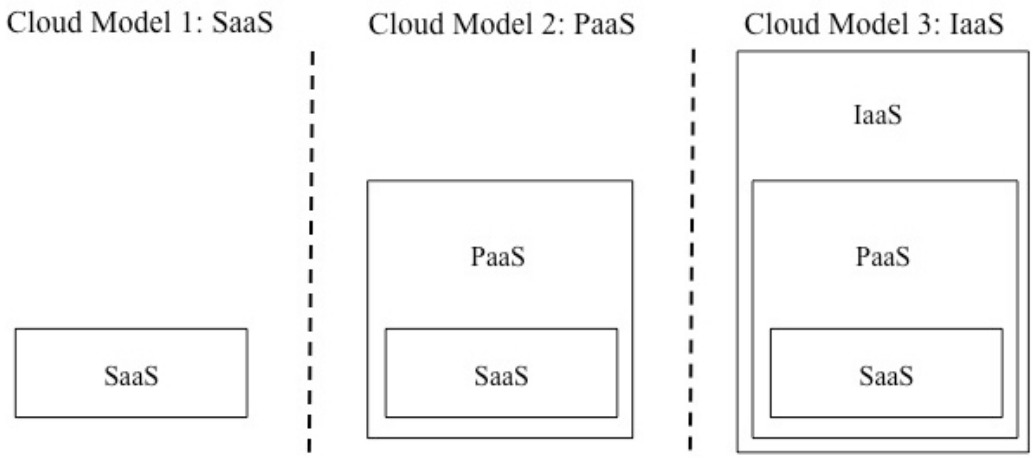
Platform as a service (PaaS) is a set of supporting tools and services designed to support programmers in the development and deployment of applications hosted on a cloud. Examples include Google's App Engine. Infrastructure as a service (IaaS) refers to the the provision of bare virtual machines or the underlying server hardware, along with the required operating system and associated networking infrastructure. With Amazon Web Services for example, a customer can request a machine instance, of a particular specification, and with their operating system of choice. It will take on average less than a minute for the virtual machine to be deployed. A customer can then lease static IP addresses, which can be mapped to their virtual machines to provide an openly addressable computer system. Another popular model is data as a service (DaaS) which includes cloud storage offerings such as Amazon's S3 and DropBox which both allow access to cloud storage via both PC and mobile. However DaaS could be considered a sub type of IaaS.

Of the three above, PaaS and IaaS are directly relevant to organizations seeking to outsource their platform or infrastructure. One concern raised by choosing PaaS relates to lock-in as developers usually have to develop their application for a particular platform. To date there has been limited interoperability between the cloud computing infrastructure of different service providers. For SaaS this isn't a significant concern given users are generally consumers of the service rather than deployers. However for PaaS it can result in signifcant barriers for a development team to port an application from one particular platform to another as often PaaS requires the use of particular languages and the deployer is abstracted from considerable detail regarding the implementation of scalability and fault tolerance.

With IaaS developers have almost full control over the virtual machines used, their geographic location, the operating system and the middleware and persistence offered. While IaaS provides more control it also requires the specialist development skills to provision,

dimension and configure the underlying components of the infrastructure. With PaaS most of this complexity is abstracted away from the end-user opening up opportunities for those with more limited technical capabilities. The diagram below illustrates the relative positioning of each of these service models. A SaaS offering could be provided through the use of PaaS or a company could develop its own PaaS and host it on third party infrastructure leased as a service.

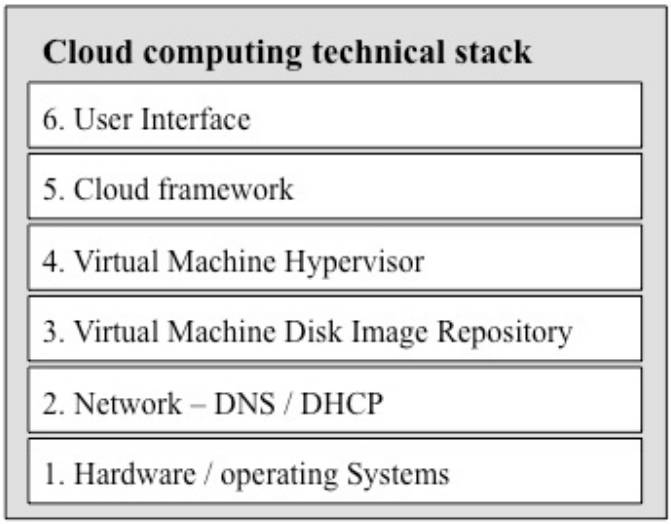
Diagram 3: Cloud service models



2.1.1. Cloud technical stack

Having defined cloud computing and considered the different service models that can apply it's worthwhile to evaluate the technical stack that is required to implement a cloud computing platform. This will help provide further insight into the potentially fundamental differences between a traditional cloud and a mobile cloud later in the analysis. The diagram below has been developed with reference to a technical discussion in the literature (Sempolinski and Thain 2010).

Diagram 4: Cloud technical stack



At the lowest layer resides the hardware and operating systems of the underlying physical servers used (often commodity hardware). These commodity servers need to have hardware extensions to support pure virtualization. Without these extensions only paravirtualization can be implemented. With pure virtualization, a guest operating system runs unmodified by a hypervisor. While a cloud computing platform can be implemented using paravirtualization, it can limit the flexibility in the software components that could be deployed and the speed of the virtual machines.

The second layer relates to the networking of the servers including the DNS, DHCP and the subnet organization of the physical machines. This includes the networking that is required to support the virtual machines having a unique MAC and IP address. In a virtualized environment the number of IP addresses used grows exponentially. A datacenter with 100,000 physical servers, each with 16 cores, could host 3.2 million virtual machines. But IPv4 addresses are limited in total to 2^{32} or just over 4 billion address spaces. Operators of cloud computing infrastructure are therefore likely to be amongst the first to switch to IPv6, which offers an address space in the trillions (IETF n.d.).

The next component is the virtual machine disk image repository. It would be impractical to complete a full install of an operating system each time a virtual machine is instantiated. Therefore template disk images with a particular OS configuration and often with other software preconfigured (a LAMP configuration for example) are stored to allow for the expedious deployment of virtual machines. The virtual machine hypervisor provides a framework that allow virtual machines to run. Examples of hypervisors include VMware, Xen and KVM. Both Xen and KVM are open source hypervisors. The cloud framework itself (such as open source platforms Eucalyptus, Nimbus and OpenNebula) manages requests from the user interface (usually web based), retrieves the requested virtual machine disk images and signals to the hypervisor to set up a virtual machine based on this specification image.

The hypervisor is the virtual machine manager (often referred to as a VMM) and allows multiple operating systems to be run concurrently on one physical server. The term hypervisor was coined by IBM in 1965, referring to software that accompanied the IBM RPQ for the 360/65 and which allowed the model to share its memory and act both as an IBM 360 and an emulated IBM 7080 (Hendricks and Hartmann 1979). A hypervisor is installed on the server hardware and interfaces between the hardware and the guest operating systems. With paravirtualization, the hypervisor can be less complex, as certain tasks can be relocated to the host server rather than being executed in virtual machines running within the host. Xen (P. Barham, K. F. Dragovic, et al. 2003), Vmware and KVM are popular hypervisors.

The cloud framework resides in the next layer of the technical stack. The framework used will depend on whether the cloud is private, public or community based. Several open source cloud frameworks are now also available. These are briefly discussed in the next section.

The last layer is the user interface. Given that cloud resources should be capable of being managed on a self-service basis it is critical to provide a provisioning portal that allows the user to configure the cloud resources they are seeking to consume with ease. Amazon provide

an online portal where users can log-in and launch and manage virtual machines. A separate set of APIs is also provided to allow for resources to be managed programmatically.

2.1.2. Frameworks for cloud participation

Given the ultimate objective of the dissertation is to explore how mobile nodes can participate in the cloud it is worth briefly considering how participants other than the large datacenter operators can participate in contributing to traditional cloud computing resources. This section will briefly review open source cloud computing frameworks that exist.

Eucalyptus is a cloud computing platform, designed to provide an open-sourced equivalent to Amazon's EC2 cloud and developed at Berkeley (Nurmi, et al. 2008). Eucalyptus implements a hypervisor agnostic operating system design. It has a modular design, with each module represented by well defined APIs, to allow researchers to use modules with other cloud computing solutions. The system allows users to start, access and terminate their virtual machines using an emulation of Amazon's EC2 SOAP interface. The framework supports virtual machines managed by the XEN hypervisor but plans to incorporate support for other hypervisors such as VMware. Each high level system component is made available as a standalone Web service. This ensures that APIs are well defined in the form of a WSDL document (Web Services Description Language). There are four high level components that comprise a Eucalyptus installation. These include:

1. Node controller: The node controller manages the execution and termination of virtual machines on the physical servers on which it runs.

2. Cluster controller: Gathers information to schedule virtual machine execution on individual node controllers. The cluster controller also manages the virtual instance networking.

3. Storage controller (Walrus): Offers a put / get API similar to Amazon's S3 storage interface. This provides a mechanism for storing and accessing data and provides the capability to have a virtual machine disk image repository.

4. Cloud controller: Provides the interface for end users and administrators. It provides the logic to query the other controllers for information relating to the available and consumed resources. It also makes high level scheduling decisions which are implemented via requests to the cluster controllers.

The developers of Eucalytus note that one of the most interesting challenges encountered in designing a cloud computing platform is the networking of the virtual machines. The networking on virtual instances must address connectivity, isolation and performance. As users are granted super user access to their virtual machines, they also have super user access to the underlying network interfaces. As a result it is important to be able to isolate a users virtual machines from the virtual machines of others. Eucalyptus works best when each cluster is placed on its own subnet, with its own reserved address range. The framework was designed with a focus on providing a private cloud computing framework for corporate enterprises. There is a strong separation between user-space and administrator-space.

OpenNebula is also focused on providing a framework for a purely private cloud that could be utilized by enterprises (Open Nebula Home Page n.d.). In order to launch a virtual machine, a user must provide a configuration file with a set of parameters that are provided to the hypervisor command line. While more tedious, this allows significant flexibility in defining the processor, memory, disk space and networking resources for the virtual machines. OpenNebula is geared towards those with a keen interest in cloud computing who want to experiment and is ideal for use with a small number of virtual machines.

Nimbus is a cloud framework geared toward the scientific community that may be less interested in the technical internals of the cloud computing system but have greater requirements for a customized cloud (Nimbus Home Page n.d.).

2.2. Summary

While cloud computing has recently witnessed popularity and exponential growth, the underlying drivers behind its emergence have been gathering pace since the late 1960s. These include the development of virtualisation technologies to share mainframe resources, rapid commoditisation of server hardware driven by Moore's law and significant increases in network speeds driven by improvements in the field of photonics. Throughout the 1980s and 1990s significant inefficiencies existed with server hardware being inefficiently utilised as organisations invested in large blocks of computational capacity with physical demarcation of critical applications. Cloud computing resolves many of the inefficiencies by providing flexibility and rapid elasticity for organisations and allowing for specialist providers to focus on the provision of highly efficient utility-scale infrastructure. The early successes of the cloud paradigm have led to a virtuous cycle of providers constructing ever-larger datacenters to optimise economies of scale, minimising overheads such as power and personnel and rationalising older generation facilities.

This section provided a clear overview of cloud service models. The typical technical stack for the cloud was then considered, followed by a high level overview of progress towards open source frameworks for allowing wider participation in the traditional cloud. It highlights that once a datacenter has been designed and deployed, the associated servers and networking are utilised across the lifetime of the assets, providing a fairly stable operating environment. The next section will consider how a cloud incorporating mobile nodes might differ.

3. THE IMPLICATION OF MOBILE NODES

This section of the dissertation will explore in detail the implications of mobile server nodes participating in a cloud computing environment. Thus far this dissertation has investigated the origins of cloud-computing, the service models, the technical stack and some of the available cloud frameworks. Is a mobile cloud different? It is important to note that the literature appears to misuse the term "mobile cloud". It usually refers to mobile phones acting as clients to leverage cloud resources (Klein, et al. 2010), (Simoens, et al. 2011), (Liang, et al. 2011). In contrast, this dissertation focuses on the options to allow mobile phones to contribute as serving nodes within a cloud infrastructure. After all if mobile devices are simply accessing traditional cloud resources, then the cloud is not mobile, but rather mobile clients are accessing a traditional cloud. To consider how mobile devices might participate, it is important to understand how mobile devices that support computation and their associated networking differ from a typical datacenter architecture. These differences can be compared across a range of dimensions including stability of operating environment, processing, power and mobility.

3.1. Stability of operating environment

The current topology of datacenters is based largely on around 40 servers mounted in a rack that are interconnected to a cluster of racks via a top-of-rack switch. These racks use 10Gb Ethernet networks with high-speed routers connecting a cluster that often contain around 45 racks. While the commodity servers within these datacenters are highly distributed, they do share some centralised resources. If, for example, a power distribution unit fails it could disable 500 to 1,000 commodity servers for several hours. More recently, clusters are often

enclosed in lorry-style self-contained units with independent fire suppression. These containers can each house approximately 2,000 servers. This topology provides significant logistical advantages for setting up a new datacenter or adding capacity to an existing one as much of the networking is preconfigured such that only the container cluster needs to be connected to the wider datacenter network. Modern datacenters are composed of a high volume of commodity servers, with a standard hardware specification, using the same operating systems and stable high-speed networking.

Fixed network infrastructure uses physical links to support consistent network bandwidth. In contrast, wireless networking is characterised by variable transmission speeds and intermittent connectivity. Mobile broadband networks have higher network latency than fixed broadband. Mobile networking has only recently evolved from being voice centric to supporting reasonable data speeds. Data rates have increased from 9.6 kilobits per second (kbps) for the Global System for Mobile Communications (GSM) standard to over 200 kbps for the General Packet Radio Service (GPRS) standard to several megabits per second with the Universal Mobile Telecommunications System (UMTS) standard using High-Speed Downlink Packet Access (HSDPA). The faster the speed of a data network, the easier it is to offload data or consume data and resources from other nodes that are interconnected. Bandwidth on mobile networks is more erratic as data transmitted is conveyed across wireless cells incorporating numerous masts and microwave backhaul links, each of which may be experiencing different levels of network load.

Above the physical network layer, fixed and wireless networks both use reliable Internet transport protocols including the Transport Control Protocol (TCP) or the unreliable, albeit faster, User Datagram Protocols (UDP). TCP was developed in an era when data communication was predominantly across fixed-line networks. Over the last decade there has been an exponential increase in the use of wireless networks for Internet access, a trend set to continue with the emergence of powerful smart mobile phones and the deployment of next

generation mobile techologies such as WiMax and LTE (McQueen 2009). Due to the network topology, wireless networks are characterised by a higher bit error rate (BER) than fixed networks. A wireless network, for example, could have a HSDPA cell deployment allowing for a maximum uncontended bandwidth of 7.2 Mbps, backhauled using one 2Mbps leased line or a microwave link to an aggregation point where several such cells contend for backhaul capacity into the core network. In the mobile access network, the propagation of the wireless broadband signal also decreases the farther the user is from the wireless mast. The architecture of a fixed network in contrast allows for data traffic to be offloaded to high bandwidth links close to the customer, mainly at a local exchange and increasingly at the local cabinet. An ADSL2+ DSLAM with 768 ports will usually have a 1 Gbps backhaul to the core IP network. The bandwidth throughput in fixed line networks is therefore much greater and less erratic than for wireless networks and contention rates are lower. Future fourth generation mobile networks that are pure IP-based such as LTE (Long Term Evolution), should increase mobile broadband speeds considerably. However, mobile networks wil always be constrained by the limitations arising from the scarce radio spectrum upon which they rely (Kennington, Olinick and Rajan 2011).

Given that TCP's original design was optimized for fixed-line networks, some deficiencies arise in wireless networks including:

- TCP assumes that packet loss is due to network congestion as against corrupted bits.

- Some underlying stability in the round trip times (RTT) for receiving acknowledgement packets are assumed as acknowledgement clocking relies on relative stability between the RTT intervals.

- TCP uses a minimum of 20 bytes of IP header and 20 bytes of TCP header per TCP packet. For low bandwidth wireless networks this is a high overhead to bear.

As wireless networks can be characterised by short-lived congestion, TCP will overreact to temporal congestion. This can result in adverse oscillations in transmission speeds and in transmission inefficiency (Tian, Xu and Ansari 2005). TCP uses a slow start algorithm to ramp up transmission rates, and significant delays in data transfer can arise if this slow start algorithm is re-initialized repeatedly due to intermittent bandwidth on a wireless network. In addition, for networks that exhibit latency the RTT can result in the algorithm being very slow in reaching its maximum transmission rates. In such situations the cost of reliability can be high, as the minimum time for a TCP transaction will always be two RTTs due to the requirement for both the sending process and the receiving process to be satisfied that acknowledgements have been received.

Given the reliability of fixed networks and the lower bit error rate (BER) relative to wireless, the design of TCP assumed that any packet loss was the result of network congestion rather than corrupted bits. There are two main issues, therefore, that arise with data transmission using TCP in mobile networks. Any packet loss gets interpreted as network congestion and the slow start algorithm therefore kicks in. Short intermittency of connections on mobile networks can result in longer TCP disconnections because of the TCP back-off mechanism.

A final distinction between fixed and mobile networks is the addressability of nodes. It is relatively easy to configure a hardware server or even a virtual machine within a typical cloud with static IP addresses. These addresses are key to enable applications to communicate with other servers. Such addressing typically requires a unique IP address and port number. The current addressing is largely based on the IPv4 standard (RFC 791 1981), which uses 32-bit addressing. This results in the address space being limited to 2^{32} addresses or just over 4 billion. This has resulted in a scarcity of IP addresses for all devices that need them. As a result most mobile networks and many Internet Service Providers use the Dynamic Host Configuration Protocol (DHCP), which allows numerous users to share a pool of IP

addresses. The impact of this is that most mobile devices do not have static IP addresses, which could be a severe limitation when seeking to use mobile devices as server nodes. This does not cause any issue where mobile devices are acting as clients of cloud services, as the cloud servers they are connected to will have static IP addresses. This address limitation will be overcome with the introduction of IPv6, as the address space of 128-bits provides 2^{128} or approximately 340 undecillion addresses[5]. Cloud computing providers are likely to be some to the first to migrate to IPv6 as the proliferation of virtual machines all require their own IP address. Over time a complete move to IPv6 would allow each mobile device to have its own static IP address. In the short term, the addressability constraint associated with the use of dynamic IP addresses on mobile devices raises problems for developing a mobile cloud.

A key consideration in designing and implementing a mobile cloud is therefore the higher likelihood of network intermittency and bit error rates. There are several practical implications for the design of a mobile cloud:

1. Mobile serving nodes are not suited to transferring large files given the higher probability of a network failure during transmission.

2. Implementing fault tolerance will require a greater number of mobile nodes than is likely to be required in a more stable fixed network. An efficient replication mechanism will be important.

3. The design of mobile cloud middleware should use UDP where possible to avoid the three-way handshake overhead associated with TCP or, where message reliability is critical, aggregate messages to the extent possible such that several messages can be transmitted simultaneously across one TCP connection.

[5] An undecillion is 10^{36}.

3.1.1. Processor, operating system and power issues

Many mobile phones use different operating systems. The three most common operating systems in use today are Apple's iOS, Google's Android and Symbian, used by Nokia, Sony, Lenovo and other device manufacturers. The Android OS is based on the Linux kernal and uses core java libraries with a Dalvik virtual machine. A subsequent benefit is that Android applications are mainly developed using standard Java. But it is important to consider that a mobile cloud, unlike a traditional datacenter is likely to consist of mobile devices with both heterogeneous operating systems and varying versions of the operating system.

Mobile devices use a different processing chip architecture than desktop, laptop or server hardware do. This processor architecture, referred to as RISC (Reduced Instruction Set Computing), is based on the concept that simplified instructions will provide higher performance if the simplification fosters higher speed execution of each instruction (Patterson and Ditzel 1980). Given that considerable time is spent in executing simple tasks to solve larger problems, the RISC architecture focus on executing these simple instructions as fast as possible. Following the design of processors using the RISC architecture, the standard architecture used in most computers became known as CISC (Complex Instruction Set Computing). The CISC architecture is also commonly referred to as x86 as most applications at the time were written for the large installed base of computers using the x86 architecture. As a result of Intel's massive investment in x86 development, the RISC architecture never enjoyed scale in the personal computing market. As chip fabrication techniques improved exponentially in line with Moore's law, and architectural improvements in design allowed for smaller chips the highest performing CPUs using both the RISC and CISC architecture had converged by 2000 (Carter 2002). The benefits of the simplified instruction set however remained significant for mobile devices. While the hardware translation overhead from x86 instructions into RISC operations was of limited concern for larger mains powered devices

such as servers, it is considered a significant overhead for mobile and embedded devices. CISC is unsuitable for mobile and embedded devices as the power consumption and heat dissipation is too high. As a result the RISC architecture is predominant in smart mobile phones, tablet computers and some netbooks. This difference between the processor architectures affects the interoperability of instruction processing. Often only the lowest layer of the operating system kernel needs to be changed if the kernel has been designed in a manner that presents a well-defined abstraction from the underlying architecture. So it should be possible to abstract middleware or end user applications from the underlying chip architecture. This would be important to ensure mobile nodes such as laptops and smartphones could participate in a mobile cloud irrespective of significant hardware differences. In many cases RISC processors will not support virtualization, limiting the ability to deploy virtual machines on mobile devices.

Operators of datacenter infrastructure seek to deploy infrastructure in a manner that will minimise the power overhead associated with operating servers, often focusing on a key energy performance metric, known as power usage effectiveness (PUE). A value of 2.0 would indicate that for every kw/h of electricity consumed by the servers, an additional kw/h is required for cooling and operating other related supporting infrastructure. The PUE is often optimised using technologies such as air economisation, which uses the external air temperature to cool servers. Some datacenter deployments have achieved highly efficient PUE ratings of 1.15 to 1.25. The implications are that the move towards consolidated warehouse-scale datacenters has resulted in significant operational efficiencies in operating computational infrastructure. A mobile cloud will not compete directly against the energy efficiency of a traditional datacenter. Due to their limited size and power consumption, smart phone design presents many engineering challenges. The phone display and wireless connectivity are the two largest consumers of energy on a smart phone (Carroll and Heiser 2010). The development of faster processors with a greater number of cores will require novel engineering solutions to minimise heat dissipation. The wireless networking technologies

including 3G / WiFi / Bluetooth and NFC, all consume varying levels of battery power. The power issue has serious implications for the ability of mobile devices to contribute as server nodes within a cloud computing environment. Historically these devices have not been designed to support always-on use as server nodes. In such a mode the battery of most smart phones will drain rapidly. Prior studies have already evaluated power consumption on mobile phones in a variety of use cases using a multimeter connected to the devices battery (Riva and Kangasharju 2008). The actual power consumption depends on the specific mobile device but Wi-Fi connected at full signal draws an average power consumption of 1,190 mW. The GSM radio consumes power in peaks of 450-480 mW and UMTS causes 1-W peaks of consumption for several seconds. Interestingly the same study shows that wireless communication is much more expensive than computation on mobile devices. Thus from an energy efficiency perspective it may not always be the case that offloading computation from a mobile device to a traditional cloud is optimal. This power constraint coupled with the processor and memory limitations currently rules out the potential of hosting several virtual machines on a single smart phone unless these are lightweight VMs used to support local applications that are not running concurrently. However, these constraints do not prevent a mobile device becoming a server in a personal access network, involving the aggregation and analysis of data locally. A mobile device could collect health data from an individual on an ongoing basis and upload key metrics to a centralized cloud server for storage and long-term trend analysis. A mobile device could also act as a file sharing node in a distributed network where files are infrequently accessed, or it could act as a server node to convey data collected periodically from local on-board sensors.

3.1.2. Mobile virtual machines

One of the key challenges highlighted from the background analysis on open source frameworks for cloud computing was that of virtual machine networking. This raises the

question as to whether it would be possible or practical to execute multiple virtual machines on a smartphone. There would be two requirements to support mobile virtual machines:

1. Hardware support, including a processor that supports virtualization.

2. Network support.

There is no doubt that smartphones will evolve to support virtualization at a hardware level. While operating systems such as Android spawn a new virtual machine sandbox for running applications (the Dalvik virtual machine), this is really just a thin software interpreter equivalent to the java virtual machine. The technical specification of most smart phones (processor, RAM, power) would make it difficult to practically run isolated virtual machines and an associated hypervisor. This is likely to change over the long term as the specifications improve and dedicated thin and highly efficient operating systems are developed to run VMs on mobile devices. While they currently have multiple network interfaces (GSM, wifi and bluetooth), it would be impractical to share these interfaces between multiple VMs. This could potentially change with the higher wireless bandwidths that become available with the widespread deployment of LTE. But this would also require allocating several static IP addresses to one smartphone, which the move to IPv6 would support. The table below provides the technical specification of current generation smartphones. A step change increase in memory and processor speed would be required to provide adequate performance to support multiple virtual machines on a smartphone, even if higher speed wireless networking technologies were available.

Table 1: Technical specifications of iPhone 4 versus HTC Desire

Device	iPhone 4	HTC Desire
Operating system	iOS4	Android
Processor	800 Mhz	1 Ghz
Storage	32GB	SD Card 2.0
RAM	512MB	576MB
Network	3G / WiFi	3G / WiFi
Location	Assisted GPS	Assisted GPS
Camera	5 megapixel	5 megapixel
Weight	4.8 ounces	4.76 ounces

Although it may therefore be theoretically possible to develop a thin hypervisor operating system that resides above the hardware layer of a smartphone, it would be of limited practical benefit for the foreseeable future. Such a virtual machine would run too slowly locally on the mobile device. However, it would be myopic to rule this option out in the future given potential advancements in the processing, storage, networking and powering of mobile devices.

3.2. Mobile phones and the cloud

The constrained processing, power and networking capabilities of current mobile devices, as discussed above, have become a driver for mobile devices leveraging the power of traditional datacenter infrastructure. Many of today's mobile applications are structured such that they are statically partitioned between the mobile device and datacenter. This results in a clear demarcation between the tasks executed on the mobile phone and those executed remotely, such as via a server hosted in a datacenter. An example of such a partition is where the mobile device runs an advanced user interface that interacts with the backend application resources in the cloud, such as Facebook's mobile application. Another example with an even lighter front end component is Google's search engine. In deciding how an application should be partitioned, the following should be considered:

1. How long it will take to process a task in a datacenter relative to on a mobile device.

2. Whether the device requires access to file resources provided elsewhere. It may be the case that the speed to which resources can be accessed across a network, rather than the processing power is the core constraint. A commodity server within a datacenter is likely to be able to retrieve data much more quickly due to its higher speed networking.

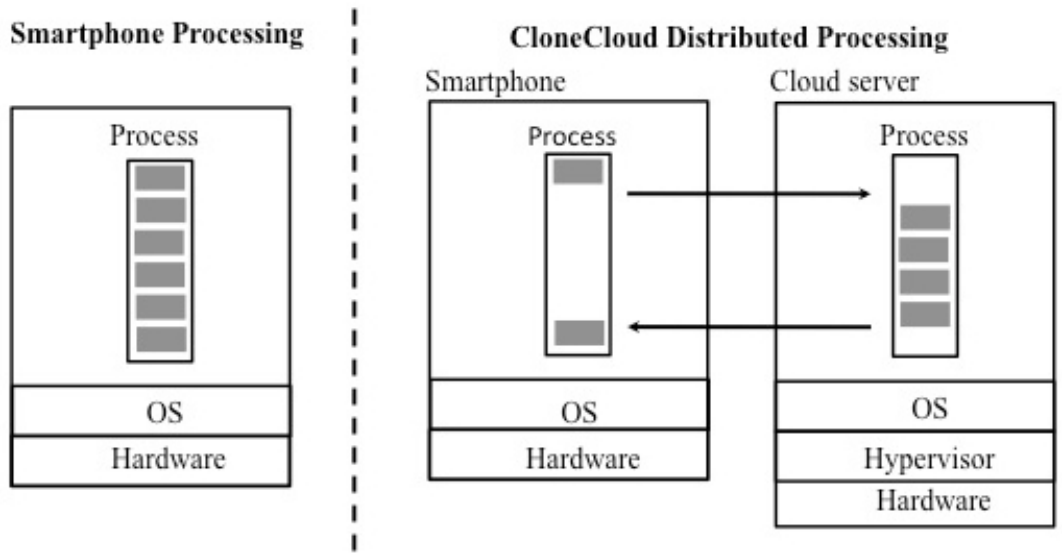
3. The latency of fixed or wireless networks. This helps to highlight the tradeoff between transmitting data for analysis and consuming processed results across a network versus local computation.

4. The manner in which the serving infrastructure is powered and the extent to which the services utilised might deplete a power source. As discussed earlier the exact tradeoff is not clear given that transmission of data might incur more energy consumption than local computation.

As most applications are statically partitioned, the demarcation between local and cloud based computation may not be optimised for the actual local resources available given the heterogeneity of technical specifications for smartphones. Some smartphones now have multicore processors, significant memory and storage and incorporate relational databases. From a network perspective, some wireless networks will have significantly greater bandwidth throughput than others, thereby decreasing the messaging and latency overhead cost associated with transmitting data to a remote server. This suggests that there may be a future opportunity to dynamically partition mobile applications between mobile computation and the datacenter computation based on the actual capabilities of the mobile devices. This notion of dyanamic partitioning has been investigated (Chun, Byung-Gon; Maniatis, Petros 2009) with researchers concluding that dynamic partitioning will become an important part of future mobile cloud computing. The same researchers have developed the concept of a supercharged virtualized clone of a smartphone that resides within a traditional datacenter (Chun, Byung-Gon; Maniatis, Petro 2010). This clone performs all of the computational

heavy lifting and transmits its results to the smartphone, thereby increasing the processing capabilities and potentially reducing the power consumption on the mobile device (depending on whether the level of processing incurs more energy than the transmission of the associated data and retrieval of results). The architecture proposed involves partially offloading execution to a clone of the smartphone hosted within a datacenter. Such an augmented execution could help superpower smartphones beyond current mobile device hardware limitations. Although the idea of offloading computation from constrained mobile devices to datacenters has been considered before, the CloneCloud approach differs in that it uses single or multiple virtualised replicas of the smartphone that reside in the cloud. The physical smartphone itself is therefore relegated to a thin client. To demonstrate the concept the researchers developed an application to scan the file system used on an Android phone. The process took 3,953 seconds on a HTC G1 phone but just 336 second on a virtualized clone residing on a Dell desktop used for testing purposes. The author has developed a high level diagram below to illustrate the architecture of CloneCloud.

Diagram 5: CloneCloud



Another benefit of cloning a smartphone is that a clone could be used to provide redundancy of the phone's data. If a smartphone were lost or stolen, a new hardware device could be synchronized with the replicas stored at the datacenter. A novel aspect of CloneCloud is that

the researchers inflate the capabilities of the virtualised smart phone such that the CPU clock rates are well in excess of the actual physical mobile device. This helps ensure that the processing can be completed quickly on data offloaded to the clone. In other similar research NTT have proposed virtual smartphones over IP, where a smartphone farm hosted in a datacenter would have a collection of virtual smartphone images, each dedicated to a particular user (Chen and Itoh 2011). Users can control their virtualized smartphone via a dedicated client installed locally on their physical smartphone. This client application receives the screen output across the network from its virtualized smartphone server instance. In this case, the Android operating system was used in conjunction with the Android-86 project (Android x86 Project 2010), allowing the Android operating system (designed for ARM processors based on the RISC processor architecture) to run on standard x86 hardware. The researchers found that it took the physical Android phone 14 seconds to open a 10Mb file, but it took just 1 second to open the same file using the virtualized smartphone in the datacenter. The virtual smartphone was also found to be 14 times faster at rendering lines and 60 times faster when drawing strings than the physical smartphone. Further analysis was completed to compare the battery consumption of local computation to that of transmitting and receiving the data to be processed to the datacenter. They found that with the same battery power an operation (in this instance resizing a jpeg image) could be performed 600 times locally or 13,800 times remotely. Such research highlights the benefits of smartphones leveraging cloud-computing infrastructure. Other research has suggested a requirement for 'Cloudlets' in which miniature cloud infrastructure is moved closer to the end user in order to reduce the round trip time (RTT) associated with interactions and therefore improve the user's experience (Satyanarayanan, et al. 2009). The researchers highlight that even with improvements in networking technologies, the delay associated with a mobile user interacting with other nodes becomes problematic if the latency is in excess of 150 milliseconds. They suggest that the ideal solution is to ensure that any cloud resources consumed should be as few network hops from the end user as possible, thus requiring the deployment of cloudlets.

The CloneCloud, virtual smartphone and cloudlet research, coupled with the current industry focus on mobile cloud computing, illustrates that the main focus has been on how to use cloud computing technology to provide more advanced processing capabilities on smartphones where the smartphone acts as a thin client and leverages the power of traditional datacenters. The table below provides a summary of the type of applications and their respective computational requirements with regard to processing power, network bandwidth and network latency thus highlighting the type of applications where a mobile might utilize traditional cloud infrastructure to enhance execution. The table suggests that applications such as Internet search, high-end gaming, video streaming and augmented reality would benefit from the computational support that can be provided by completing a significant proportion of the processing in a traditional datacenter.

Table 2: Taxonomy of applications and associated computing requirements

	Computing Requirements		
Application	**Processing Power**	**Network Bandwidth**	**Network Latency**
Internet search	High	Low	High
Email	Low	Low	High
Web Browsing	Low	Low	High
Social Networking	Low	Low	Medium
Gaming	High	Medium	Low
Video Streaming	High	High	Low
Augmented Reality	High	High	Low
File sharing	Low	Depends on content	Medium

This raises the question as to the practical benefits of having the processing or storage pushed to the edge of the network as a mobile cloud would do. The table above suggests the use cases are different and that applications requiring the dissemination and collaboration on files of a reasonable size would fall under a mobile cloud umbrella. Mobile nodes may have particular advantages in allowing their owners to maintain physical control over the hosting of their services or in allowing collection and collation of data across large geographies.

3.3. Summary

This section compared a traditional cloud arhitecture to one incorporating mobile server nodes. It highlighted that traditional datacenters enjoy a relatively stable operating environment compared to that of a mobile cloud. From a hardware perspective, mobile nodes have less processing power than commodity servers and are highly energy constrained due to being battery powered. Smartphones were never designed to be used as always-on serving infrastructure. The heterogeneity of mobile hardware and operating systems is also a concern. Given these differences, a mobile cloud will exhibit different failure semantics than a traditional cloud, especially with regard to availability and performance.

From a networking perspective wireless networks suffer from lower bandwidth and greater intermittency resulting in a degradation of the performance of transmission protocols. TCP overreacts to temporal congestion, and its large packet headings are a high overhead when transmitting maintenance messages within a mobile cloud. The use of UDP for certain message types (such as mobile nodes transmitting check-in messages) could partly alleviate these shortcomings in a wireless environment, such as the requirement for TCP's three-way handshake. Another alternative is to aggregate and transmit messages simultaneously to allow for a more efficient use of a TCP connection. This may not always be possible and would require efficient scheduling of such messages. Though wireless network speeds will improve with the deployment of technologies such as WiMax and LTE, the associated speeds will not parallel fixed networks due to the topology of the networks and the scarce radio spectrum upon which wireless networks rely.

As a consequence of the network and power issues that arise with mobile nodes, the ability to effectively federate such mobile resources will require an architecture that solves the unique availability challenge. This could be accomplished through the replication of state or content

across a number of nodes. A tradeoff is required to balance the higher overhead associated with such replication with the availability and performance of the network. It will therefore be important to develop an efficient replication mechanism in order to limit this overhead, which can also be tuned to support various applications that could reside on a mobile cloud.

The CloneCloud and NTT research reinforced the widely held perception that the future will focus on smartphones leveraging traditional cloud resources for heavyweight computation. The taxonomy of applications also demonstrated that certain types of low latency and high computation applications, such as high-end gaming are unlikely to suit a distributed mobile cloud in the medium term. However a mobile cloud could be effective in the aggregation and dissemination of information and in providing a means whereby owners could maintain physical control over the services they host. The table summarises the analysis into a comparison of the traditional and the potential mobile cloud.

Table 3: Comparison of traditional versus mobile cloud

Cloud type	**Traditional Cloud**	**Mobile Cloud**
Processing	High specification	Low specification
Software environment	Homogenous OS	Heterogeneous OS
Hardware heterogeneity	Standard	Heterogeneous
Geographic distribution	Clustered, static	Dispersed, mobile
Network	High speed, low latency	Low speed, high latency
Network connectivity	Always on	Intermittent
Network environment	Highly controlled	Low control
Power	Mains / UPS / Generator	Battery

4. OPTIONS FOR IMPLEMENTING A MOBILE CLOUD

This section considers the options that exist for implementing a mobile cloud and the challenges that arise with each. The objective is to determine an option suitable for implementation.

4.1. Option 1: Distributed processing

The background analysis on cloud computing uncovered a trend starting in the late 1960s with the implementation of virtualization to allocate resources on high power mainframes to the development of warehouse scale facilities housing highly networked servers in modern datacenters today. This move to highly distributed computing has been supported by the provision of software abstractions that make it easier for developers to write code that can run on a network of commodity servers. A mobile cloud could be used for distributed processing whereby all of the participating nodes process discrete tasks that contribute towards solving a larger problem. It is therefore worthwhile to consider the option of a distributed mobile processing cloud.

In the past tuple spaces have been proposed for distributed processing. In mathematics and computer science, a tuple is an ordered list of elements. A tuple space provides a repository of tuples that can be accessed concurrently and allow for the shared processing of data. Tuple spaces were the theoretical underpinning of a language called Linda, composed of workers and a shared tuple space memory. Linda was an invention of the Yale Linda Group led by David Gelernter. Workers can exchange information and synchronize within the tuple space. JavaSpaces provides a Java implementation of Linda that was incorporated in the Jini project. A distributed implementation of Linda (Xu and Liskov 1989) was developed, allowing heterogeneous uniprocessor computers to run large processing jobs in parallel instead of

requiring multi-processor machines. Fault tolerance can be added by replicating tuple spaces across several nodes (Kambhatla and Walpole 1990). Each worker interacts with a tuple space using three types of operations: out, in and read. Out(x) adds the tuple x to the tuple space. In and read are used to extract information from a matching tuple in a tuple space. Operations on a replicated tuple space are implemented as follows for an operation x.

1. The out(x) operation will write to all replicas. The request to execute this operation is broadcast to all replicas and the worker waits for an acknowledgement from the replicas.

2. At each replica, x is stored in the local tuple space, and an acknowledgement is sent to the worker.

3. The in operation removes the same tuple from each replica. First it acquires the lock and reads from the replica (referred to as the In1 phase). If the tuple is locked by another worker, this request is refused. In the in2 phase the tuple is removed from the tuble space.

Linda has been further extended with Lime, Linda in a Mobile Environment (Murphy, Picco and Roman 2001). The characteristics of Linda were seen to resonate in a mobile environment as the use of distributed tuple spaces allows communication to be decoupled in time and space. Lime provides a middleware abstraction for the development of applications exhibiting physical or logical mobility of hosts. With Lime the tuple space is broken into many smaller tuple spaces and each is then associated with a mobile node. The tuple space on each mobile node can only access the global tuple space via an interface tuple space (ITS). The ITS contain tuples that the mobile node will make available to other mobile nodes. The access to the ITS takes place using the primitives set out with Linda. The resources available therefore change dynamically as tasks are processed on co-located mobile nodes. When a new mobile node arrives, tuples in the ITS of the new mobile node are merged with those shared from the

other mobile nodes. Similarly when a mobile node leaves, the corresponding data is no longer visible to remaining mobile nodes through their ITSs.

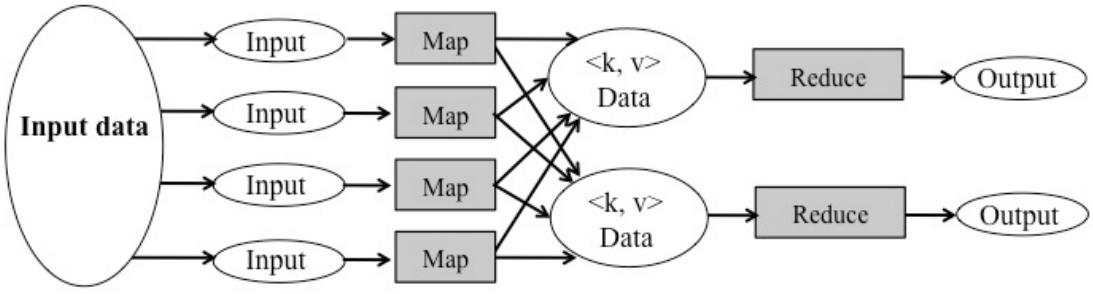
MapReduce provides another distributed processing framework. Google patented MapReduce in 2004 as a system and method for efficient large-scale data processing. MapReduce libraries have been written in C++, Java, Python, Perl, Ruby and Erlang. The Map and Reduce functions were derived from functional language primitives. The process consists of two steps:

1. Map: A master node partitions the input into smaller problems and distributes these to worker nodes. A worker node can decide to further partition the problem and allocate it to nodes under its control. The worker processes the problem, and transmits the answer back to its master node. The map function is therefore applied and produces intermediary key / value pairs.

2. Reduce: The key / value pairs within a partition are passed into a reducer function. The master node then combines the answers from all worker nodes to complete the overall analysis.

The diagram below illustrates how MapReduce works.

Diagram 6: MapReduce



If one mapper or reducer fails, the work can be rescheduled so long as the input data is still available. In theory, therefore, the framework supports an environment with high levels of

node churn, as would be the case with mobile nodes using wireless networks. The Map and Reduce functions are both defined in a data structure of key / value pairs. Every node in the network must report back periodically to either confirm that work has been completed or to provide a status update. If a node fails to report, it is presumed to have failed and the work is assigned to other nodes. Google utilise MapReduce for web-link graph reversal and web access log statistics. However, it has also been used for grid computing on desktops and in mobile systems.

Hadoop provides an open source MapReduce framework built using java. Hadoop is targeted at a datacenter environment with long running applications and extensively uses XML, which incurs a high overhead to parse. Hadoop is also usually deployed on commodity servers interconnected at a minimum of 1 Gbit/s. This is required as Hadoop is designed to support batch processing using 64 MB blocks. MapReduce or Hadoop would therefore need to be tailored to support the much lower block sizes and limited network connectivity required for mobile nodes (Marinelli 2009). MapReduce has been deployed to a mobile testbed (Dou, et al. 2010). The authors proposed Misco, a MapReduce framework for mobile devices and personal computers. The testbed was implemented using Nokia N95 smartpones. Misco comprised a master server and a number of worker nodes. The Misco server maintains the input, intermediary and result data associated with the applications and keeps track of worker progress. The workers were designed to use a polling approach such that the worker polls the server each time it becomes available. The researchers noted that if the polling frequency is too short, battery power is wasted whereas if the polling rate is too long it is difficult to align the task arrival rate with working nodes efficiently. Misco is implemented using Python, allowing the worker code to be implemented on many different mobile platforms. Misco could be extended to allow for the worker nodes to collect data and implement a MapReduce function on this data rather than only process data provided by the master server. This could be of use for processing data collected via sensors onboard a smartphone prior to transmission in a distributed mobile processing cloud.

The research investigating the use of tuple spaces and MapReduce for mobile environments used mobile devices dedicated to the experiments. Most smartphone operating systems are not designed to support significant levels of multithreading. For interpreted languages such as Java, which can be used in developing applications for the Android operating system, the virtual machine will add another layer of abstraction that further impedes performance. Android for example, limits the heap size of each application to 16 MB. Java manages I/O in a uniform manner whether communication is local or across a network. However, in a wireless network there will be significant differences between local I/O and I/O across the network, leading to a greater overhead in managing fault tolerance for distributed mobile processing. The latency in a wireless network could result in a requirement to re-route requests even though a node is actually available. The fact that mobile devices are generally battery constrained and are not designed for long-lasting computation or extensive multithreading suggests that a distributed mobile processing cloud would be of limited benefit and only suitable for certain applications. Such applications could include a disaster scenario where traditional processing infrastructure is unavailable or sensor applications whereby it is more efficient for the mobile node to perform initial processing of data rather than transmitting raw data that could consume excessive battery power.

4.2. Option 2: Distributed mobile storage system

The concept for a distributed mobile storage system was published around 2002 (Sobti, et al. 2002). Such a system would have to address a number of challenges, including the ability to search and retrieve a file and how to ensure consistency across multiple mobile devices given different versions of files. The authors approached the first problem using a location and topology-sensitive multicasting solution. To resolve the consistency problem they used lazy

peer-to-peer propagation of invalidation information. At the time the authors proposed a device, referred to as a Skunk, consisting of a processor, a storage element, an ad hoc network connectivity interface and a WAN connectivity interface. Today smartphones and mobile broadband ensure no separate devices are required. Availability in the presence of node failure is a key challenge to be addressed. An effective replication policy is important.

4.2.1. Replication

Ensuring the availability of data in the presence of node failure is generally achieved by means of replication. This is illustrated in the table below when three replicas are placed on servers, each with 99% availability. The probability of one node failing is 1/100. But with the addition of two replicas the probability becomes 1-(1/100 x 1/100 x 1/100) = 1/10,000.

Table 4: The impact of replication on availability

	Availability	Days in year	Downtime hours
Node availability x	99%	365	3.65000
2x replication	99.99%	365	0.03650
3x replication	99.9999%	365	0.00036

But replication introduces tradeoffs between availability and service consistency (Hennessey 1999). Service consistency guarantees that concurrent updates will not conflict but will limit system availability as the consistency protocols usually require synchronous access to at least a subset of all replicas, in order to ensure a uniform view of write ordering. If any of the replicas in the required subset cannot be reached, the entire service will be unavailable. There has been extensive research into how to optimise the balance between availability and service consistency (K. Peterson, M. Spreitzer, et al. 1997). The Coda file system (Kistler and Satyanarayanan 1992) uses optimistic replication where replicas are allowed to diverge for a period of time. Here replicas will converge only when the system has been quiesced,

generally requiring flushing of any outstanding write operations. Levels of replica divergence need to bounded or the system can be left in a "delusional" state (Gray, et al. 1996).

It is worth exploring how replication might apply with mobile nodes. With the Bayou system (K. Peterson, M. Spreitzer, et al. 1997) each device has a local replica of the database. There are two states of operation: disconnected and merging. When merging all new updates to other available databases are added to the local replica. This model is often used but is entirely inappropriate for a mobile cloud as it is unlikely that a mobile node will have the capacity to hold all of the required data in its database due to resource constraints. The original Coda system (Kistler and Satyanarayanan 1992) shared Bayou's model of disconnection but was extended to support weak network connectivity as would be required for a mobile cloud (Mummert, Ebling and Satyanarayanan 1995). However Coda clearly distinguished between clients and servers and was not architected to allow peer-to-peer functionality. So while it was extended to support weak network connectivity it is not naturally suited towards a mobile cloud where each mobile node may have a client and a server collocated. Other research (Li, Reed agus Lippman 2008), analyses the design tradeoffs of a collaborative mobile storage system when a peer node tries to access a given item, and the corresponding upload and retrieval delay for a stored data object of different sizes under a variety of circumstances. A common challenge that arises in the literature relates to providing enough redundancy for the stored data objects while minimising the overall storage overhead. The most common practice for providing redundancy is by either replication or erasure code. In a typical erasure code scheme, an original data object of size S bytes is split into n data fragments and a certain mathematical transform maps n data fragments into n+m total fragments such that any n encoded fragments out of the n+m total fragments can recover the original data object. In the analysis the authors assume that the total number of mobile devices in the given P2P collaborative storage system is N. They also assume that a data object is divided into n fragments of equal sizes, which are then encoded into n+m total fragments using Reed-Solomon code.

Collaborative storage systems using mobile devices face special challenges compared with Internet-based systems. Wireless link speed is typically much lower that that of a wired counterpart. Similarly channel conditions can be erratic over time and may not hold to allow the consistent transfer of large blocks of storage. The delay to upload an object to peer nodes or to retrieve an object from peer nodes has to be reasonably small, especially for wireless networks.

4.3. Option 3: Peer-to-peer architecture

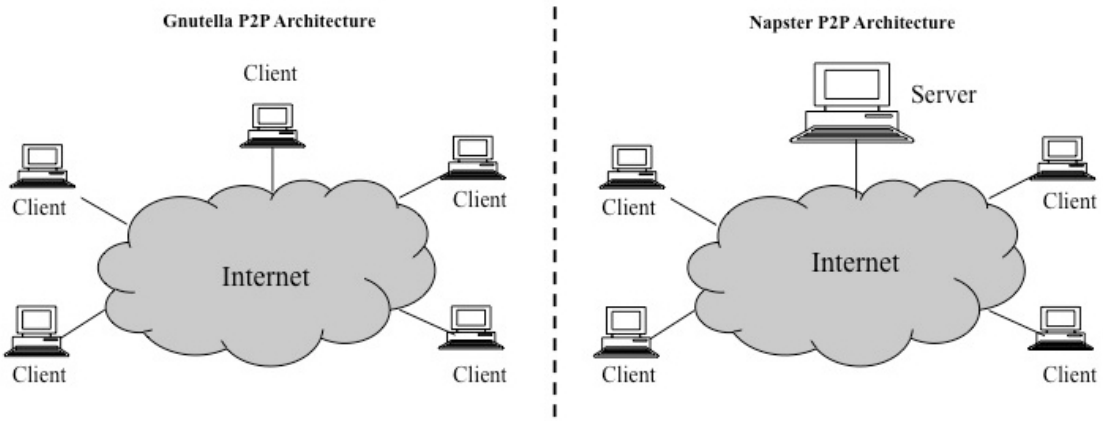
This section will investigate the potential to use a peer-to-peer (P2P) architecture in part or whole for the development of a mobile cloud. With P2P networks, participating nodes contribute a portion of their resources to the network. These participating nodes are both consumers and suppliers of the overall network resources, which may include distributed processing and storage. In P2P networks resources are located at nodes on the edge of a network. Each peer node shares autonomy and has similar rights. The provision of a P2P network utilising mobile nodes could provide an option for a mobile cloud but would require the incorporation of significant fault tolerance. P2P networks currently generate about two thirds of all traffic on Internet backbones and can be characterised as unstructured or structured networks.

4.3.1. Unstructured P2P networks

With an unstructured P2P network, no specific network planning is used to determine how resources will be either distributed or retrieved within the network. A key benefit of an unstructured P2P network is that no network planning is required. Gnutella is an example of

such a P2P network and uses message flooding to locate files by searching every node participating in the network. The Gnutella architecture could be utilised with a small number of powerful nodes. It would not scale to a large volume of nodes due to the message flooding mechanism and is not therefore a suitable architecture for supporting a mobile cloud. Gnutella does seek to improve transmission efficiency through using UDP queries for message flooding and TCP for the file transfer. Napster, the popular music-sharing network, was also an example of an unstructured P2P network, but it utilised a centralised server directory to coordinate file lookups. The Napster architecture significantly reduces the volume of messages required. A client node simply queries a directory, located at a centralised server, to determine the location of the required resources. The directory responds with the locations and address details for nodes where the required content can be found. A client can then initialise a direct peer connection to the serving node using the address information provided by the directory. The diagram below illustrates the difference between the Gnutella and Napster architecture.

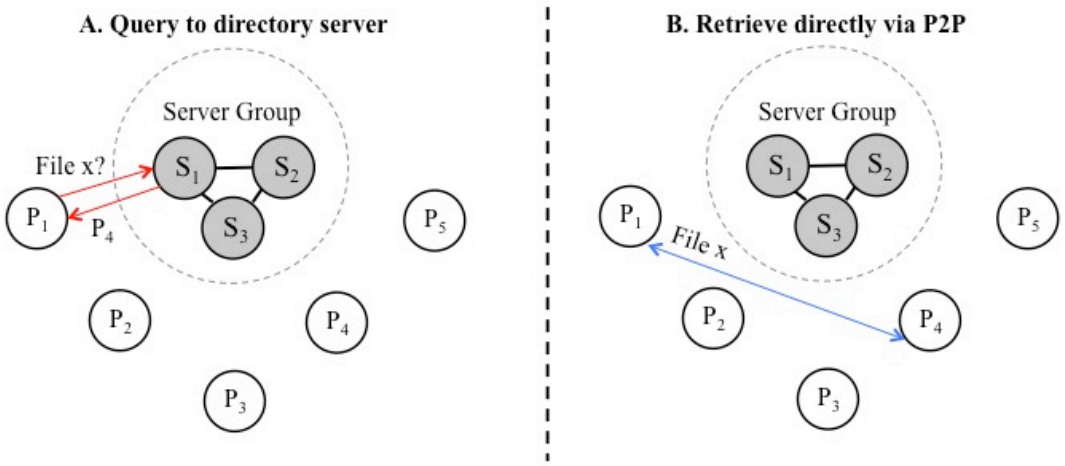
Diagram 7: Unstructured P2P architectures



The directory server in the Napster architecture is a single point of failure. This can be alleviated through the use of distributed directory servers where the directory is replicated

across redundant servers. This is illustrated in the diagram below and would require that the data residing on the servers participating within the group be synchronised periodically. This architecture would not be feasible in a mobile cloud unless a decision is made to locate the directory server and databases on traditional cloud infrastructure.

Diagram 8: P2P with distributed directory



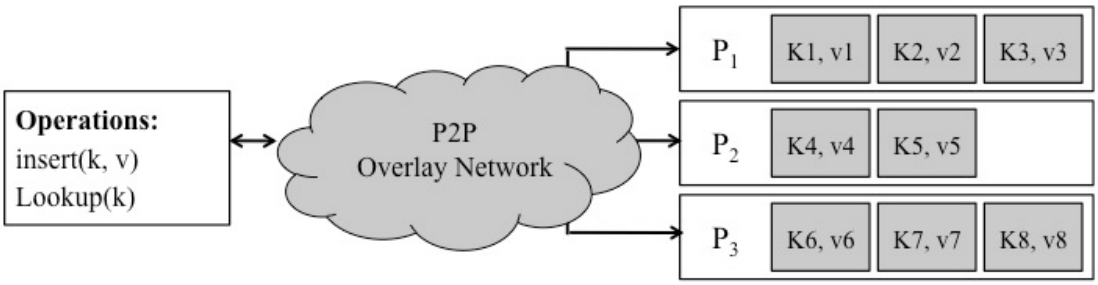
The success of Gnutella and Napster led to increased interest in P2P architectures and how to organise a network of P2P nodes that allows for node arrival and failure, load balancing and efficient routing between nodes. This led to the development of structured P2P networks.

4.3.2. Structured P2P networks

In a structured P2P network the resources are associated with particular nodes usually through a distributed hash table (DHT) that utilises consistent hashing of a given address space. A DHT can store resources at locations throughout the network and will allow for rapid file location and retrieval based on the use of exact key / value pairs. These key / value pairs

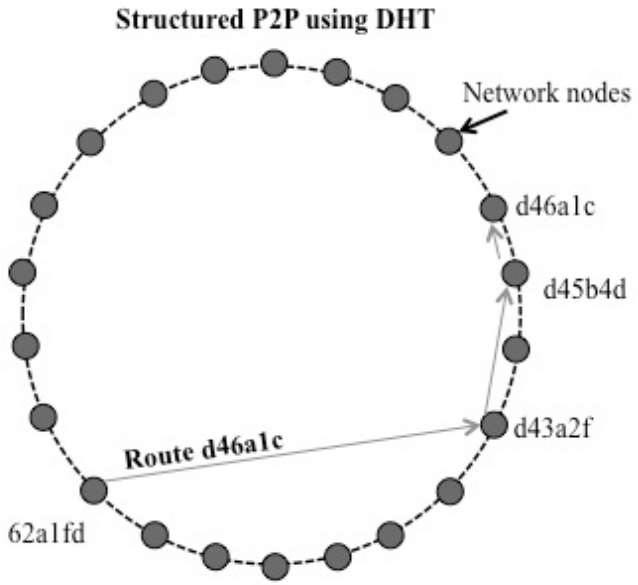
allow each participant node to search for a file or resource associated with a particular key. This architecture is illustrated in the diagram below.

Diagram 9: DHT based network



A structured P2P network is efficient for load balancing resources across a distributed network. A benefit of consistent hashing is that the removal or addition of a network node will only change the key / values owned by the adjacent nodes in the network rather than the nodes across the entire network. The node with the numerically closest node ID maintains the object. Consistent hashing supports a limited movement of objects stored within the DHT, thereby minimizing the reorganization of content required to support node churn. This characteristic of a structured P2P architecture could be useful in a mobile cloud environment. Structured P2P topologies share common properties. For a key K, a node either has a node ID that owns K or has a link to a node where the ID is close to K in the keyspace defined. This is illustrated in the diagram below where the circle represents the keyspace of the network and each node routes a request to the nearest node in its routing table to the end destination.

Diagram 10: Structured P2P architectures



There is however a trade off between minimizing the number of hops in any route (route length) to complete requests quickly and keeping the maximum number of adjacent neighbours of a node low, referred to as maximum node degree. The node degree impacts the extent of the maintenance overhead associated with nodes churning in the network. The fewer the number of neighbours the less the network needs to re-calibrate after nodes churning. However shorter hop counts across the network requires a higher maximum node degree. A balance must be made around latency of requests versus the ability of the network to be able to easily administer node churn. From an implementation perspective the most notable differences between DHTs are:

1. keyspace: Some utilise a 128 bit or 160 bit keyspace. SHA1 is often used and the key K can be a hash of the files content rather than its name so that the file can still be located if it is renamed in the network.

2. Redundancy: Needs to be added to improve reliability. For a mobile cloud the value of the key / value pair would need to be stored in multiple mobile nodes.

Pastry (Rouston and Druschel 2001) provides P2P middleware utilising a DHT overlay network. Pastry performs application level routing and object location in a potentially large overlay network of nodes connected via the Internet. It can be used to support a variety of peer-to-peer applications, including global data storage, data sharing, and group communication and naming. When presented with a message and a key, a Pastry node efficiently routes the message to the node with a node id that is numerically closest to the key, among all live Pastry nodes. Pastry takes into account network locality; seeking to minimize the distance messages travel, according to a scalar proximity metric like the number of IP routing hops. The node id is used to indicate a node's position in the logical circular node id space, which ranges from 0 to 2^{128-1}. Each node in the Pastry peer-to-peer overlay network is assigned a 128-bit node identifier (GUID). The node id is assigned randomly when a node joins the system. Consistent hashing results in node ids being generated such that the resulting set of node ids are uniformly distributed in the 128-bit node id space. As a result of this random assignment of node ids, nodes with adjacent node ids are diverse in geography, ownership, jurisdiction and network attachment. The nodes within close proximity in terms of the GUIDs may be geographically dispersed (the keyspace ring is logical not physical). Each node has a routing table that contains $\log_2^b$ N rows and each of the entries contains both the GUID and IP address for each node. The routing within the network is based on the GUID overlay rather than the IP address. A routing table for node A consists of entries where a node shares the same first n digits as node A but where the n+1 digit in the GUID differs. In practice an effort is made to ensure that the node addresses placed in the routing table are those closest to node A. It is expect that 0(log N) routing steps will be required to locate content in the network and that a routing table size of 0(log N) is also required at each node. One of the main benefits of this approach is that a Pastry node does not need to be aware of all other nodes within the network. This attribute is key in a mobile cloud, given it's a dynamic environment where nodes churn and it would be impractical to facilitate the extensive messaging required to allow each node be completely aware of the global state of the network at any given time.

Pastry nodes keep track of their immediate neighbours in the GUID space, and can notify an application of new nodes joining the network or nodes failing. A Pastry network seeks to minimize the distance each message travels by using metrics such as the route length (hops) a message traverses. The researchers emulated a network of up to 100,000 nodes to illustrate that Pastry is decentralized, scalable and self-organizing. Pastry also utilises a leaf set where each node maintains IP addresses of the node number with the L/2 numerically smallest and largest nodeIDs respectively. This allows for routing efficiency. The Pastry API is relatively simple with the following commands:

route(M, K) – route message M to ode with nodeID numerically closest to K.

deliver(M) – deliver message M to application.

Forwarding(M, K) – message M is forwarded towards Key K.

newLeaf(L) – report a change in the leaf set L to the application.

The pastry routing procedure is as follows:

if(destination is within range of leaf set)

forward to numerically closest member

else

let l = length of shared prefix

let d = value of l-th digit in D's address

if(R_1^d exists) forward to R_1^d (move further up the number hierarchy)

else

forward to known node sharing as long a prefix but numerically closer than current node.

Despite concurrent failures eventual delivery is guaranteed unless |L|/2 nodes with adjacent node ids fail simultaneously where |L| is a configuration parameter usually set at 16 or 32.

In each routing step, a node normally forwards the message to the node whose node id shares with the key a prefix that is at least one digit longer than the prefix that the key shares with the present node's id. If no such node is known, the message is forwarded to a node whose node id shares a prefix with the key as long as the current node, but where the node id is numerically closer. To support this routing, each node must maintain some routing state. Each Pastry node maintains a routing table, a neighbourhood set and a leaf set. A nodes' routing table R is organized into rows with 2^b -1 entries in each. Each entry in the routing table contains the IP address of one of potentially many nodes whose nodeID have the appropriate prefix. The uniform distribution of node ids ensures an even population of the node id space; thus on average only [$\log_{2^b} N$] rows are populated in the routing table.

Another P2P middleware solution is Chord (Stoica, et al. 2001). It is similar to Pastry but without the overlay routing network. A Chord network is also organised in a logical ring with peers assigned a key through the use of a hash function. Self-Chord (Forestiero, et al. 2010) advances Chord to be self-organizing based on the biological inspiration of ant behaviour. Self-chord decouples the naming of resources and peers, resulting in two sets of keys / indices that can have different cardinalities. Unlike Chord, Self-Chord doesn't assign keys to specific nodes in the network. Rather it focuses on the ability to re-order the keys as necessary across a network of nodes to ensure a fair distribution as the network changes. With Pastry or Chord, certain operations are required when nodes join a network or when new resources need to be published to the network. Resources are assigned to the network nodes whose indexes match the resource keys. This isn't necessary with Self-Chord because it will allow for the continuous re-ordering of keys in order to foster network scalability.

Kademlia (Maymounkov and Mazieres 2002), another P2P middleware option, specifies the structure of the network and exchange of information through node lookups. The nodes communicate using UDP. Kademlia uses SHA1 hashing with each participating node having

a node ID within a larger 160 bit key space. Like Pastry, a node ID routing algorithm allows for the efficient location of values for any given key. When searching for some value, the algorithm needs to know the associated key and explores the network in several steps. Each step will find nodes that are closer to the key until the contacted node returns the value or no closer nodes are found. Like many DHT based networks, Kademlia contacts only 0(logN) nodes during a search. A basic Kademlia network with 2^n nodes will only take n steps (in the worst case) to find that node. Kademlia has four message types including PING (to verify a node is alive), STORE (store key, value), FIND_NODE and FIND_VALUE.

A node that wishes to join a structured P2P network must first go through a bootstrap process. In this phase, the node needs to know the IP address and port of another node (obtained from the user, or from a stored list) that is already participating in the network. If the bootstrapping node has not yet participated in the network, it computes a random ID number that has not already been assigned to any other node. It uses this ID until leaving the network. The joining node inserts the bootstrap node into one of its k-buckets.

A P2P architecture provides a potential architecture for a highly distributed mobile cloud and a structured option could help avoid any requirement for centralised resources. While the current generation P2P systems can be considered an advanced distributed file system, they generally only allow for simple and exact searches to be performed using key / value pairs. One potential disadvantage of a fully distributed lookup is that each hop in a wireless network could require a message to traverse across a full mobile network when 3G / UMTS or LTE is used. Where several hops are required this could add significant message latency to any application.

The table below compares traditional cloud architecture with a distributed P2P architecture.

Table 5: Traditional cloud architecture versus P2P

	Traditional Cloud	P2P Cloud
Environment	Controlled	Uncontrolled
Churn	None	High churn
Architecture	Centralized	Distributed
Access points	Single point	Multiple point
Nodes	Homogenous	Heterogenous

The table below provides a summary of centralised and P2P network types.

Table 6: Taxonomy of P2P network architectures

P2P Networks	Centralised	Pure P2P	Hybrid P2P	DHT Based
Directory	Centralised	Distributed	Centralised	Distributed
Example	Napster	Gnutella 0.4 Freenet	Gnutella 0.6 eDonkey	Pastry. Chord, Kademlia

4.4. Option 4: Mobile Web Services

Service-oriented architecture (SOA) provides another option for implementing a mobile cloud in whole or part. Web services are based on the use of open Internet standards like WSDL, XML, REST and SOAP (Kreger 2001). These standards can be used to enable loosely coupled interoperability between applications. SOA allows developers to encapsulate application methods as services that a client can then access without any knowledge, or control over, their internal workings (Foster 2005).

The Web Services Description Language (WSDL) defines the methods and bindings, providing an API, similar to an Interface Definition Language (IDL), which allows the user to understand how to utilize the web Service. In order to invoke the underlying methods exposed as a web service, requests and responses can be sent either using REST or the Simple Object Access Protocol (SOAP). HTTP is used for the transport protocol. As Web services are based on accepted standards, solutions are language neutral. A benefit of a Web services architecture for a mobile cloud is that the system could incorporate smartphones or other mobile devices irrespective of the operating systems used, with nodes communicating across standard HTTP. However a web server of some kind is a technical prerequisite for the provision of mobile Web services. This could be achieved by installing a cut down version of Apache, the open source web server (Apache Open Source Web Server n.d.). Alternatively a simple lightweight web server could be developed given that a distributed mobile cloud would not require each node to support high-load HTTP requests. One advantage of this approach is that a mobile web service could be used to offer services that simply could not be easily replicated by traditional cloud computing infrastructure. Users could offer a set of

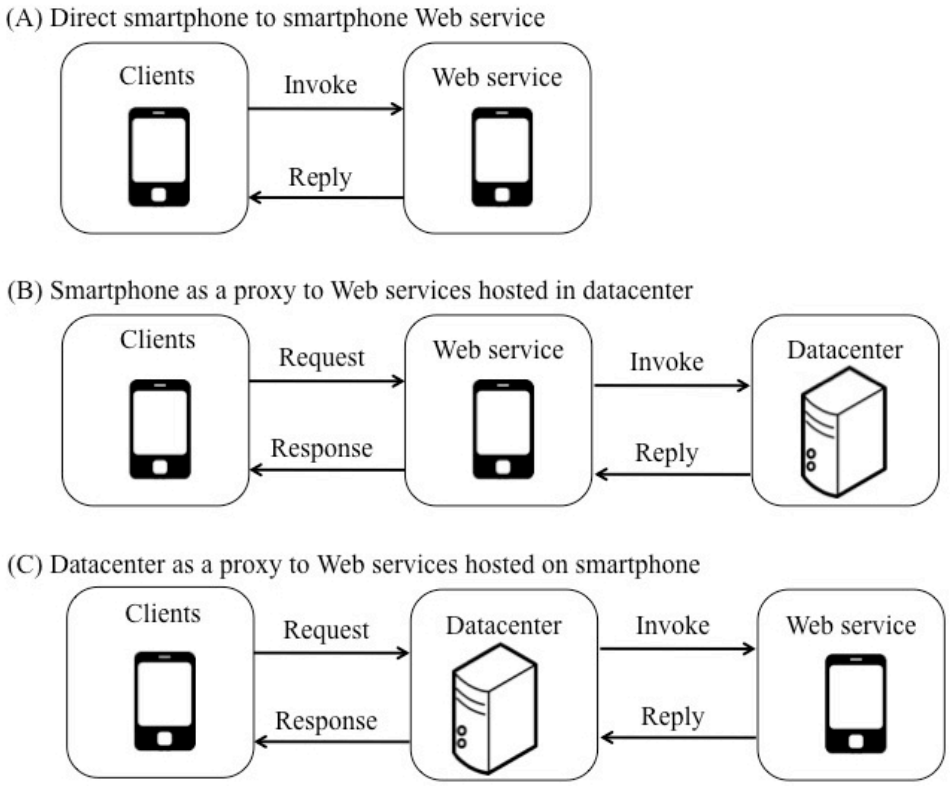
personal Web services that could be invoked by others remotely to provide useful information.

There has been limited research into the practical implementation issues and potential of hosting Web services on mobile devices. Research from Nokia (Wikman and Dosa 2006) developed the S60 mobile web server, which was effectively a cut down version of Apache. However as of January 2010, Nokia discontinued this web server. There are reasons that may explain why this initiative was unsuccessful. The IP addresses allocated to mobile phones are usually dynamically allocated. One of the reasons for this is that mobile phones are not designed to host servers for services. In a client role, the mobile device doesn't require a static IP address. To date therefore, mobile devices are not generally allocated static IP addresses, hence it could be difficult to initiate a connection to such a device as it would not be publically addressable. In addition the firewalls of mobile operators are often configured to prevent traffic initiated outside the wireless network. As a result HTTP or socket requests initiated from outside the network may not reach devices within the network even if they had a static IP address. The use of IPv6 to facilitate static IP addresses to all Internet connected devices coupled with increased demand for direct device addressability will alleviate the addressability issue in the coming years opening up the potential for mobile Web services.

IBM was the first to consider the hosting of Web services on mobile devices, through the development of a shopping kiosk application (Berger, et al. 2003). In this study issues such as service discovery, device disambiguation, software footprint and security were considered. Researchers at Macquerie University also proposed a framework for hosting Web services on mobile devices (Hassan, Zhao and Yang 2010). This research highlighted that the provision of web services from a mobile phone could be of great use in emergency or disaster situations, where skilled personnel such as doctors could be located by invoking a web service hosted via their mobile phone that returns their location using GPS. Their framework proposes that the Web service interfaces are left on the mobile devices, but heavy-duty computing tasks are

delegated to remote servers hosted in a normal datacenter. This is achieved through the mobile device acting as the integration point with the support of backend servers and remote Web services as depicted by B in the diagram below. It would also be technically feasible to reverse this, so that the Web services are accessed via a traditional SaaS user interface from a cloud, but where the actual methods invoked reside locally on a user's smartphone as depicted in C below. This could allow for mobile nodes to opt in and out of a directory of Web services that is centralised and abstract any requirement for the requesting client to have knowledge of the terminating IP addresses of the mobile devices providing the actual end service.

Diagram 11: Structured P2P architectures



The use of REST Web Services to allow mobile devices to host short-lived services has been considered (Fahad, et al. 2009). They found that the demands on a wireless network are significantly reduced with REST when compared to SOAP and that the use of REST could have positive effects in reducing the processing latencies when using a mobile server. SOAP

messages are particularly verbose resulting in a large transmission overhead and subsequently a processing overhead to parse message content. Clearly developing a system based on REST would impose a requirement on using HTTP and URLs. Due to this dependency, RESTful web Services are tightly coupled to HTTP methods. GET, PUT, POST and DELETE are the most commonly used methods. The authors highlight that the use of REST could create some subtleties in situations where a service offers several resources of one kind resulting in it becoming difficult to map methods to URLs in a manner that is reasonably self-explanatory.

Despite the fact that Web services standards exist, the end users of mobile devices are not likely to be able to personally code and configure the actual mobile Web services they wish to offer. It would be more realistic to develop software that abstracts users from the code altogether, thereby alleviating the burden of publishing mobile Web services. The analysis suggests that mobile Web services could be successfully exploited either in the provision of personalised services that are published by individuals and can be invoked by others or to provide a loosely coupled communications mechanism for sharing information in a federated mobile cloud.

4.5. Summary

The preceding three sections provided an in-depth background analysis into the emergence of cloud computing, explored the potential implications of having mobile nodes participating in the provision of services within a cloud environment and then considered some of the options that may exist for implementing a mobile cloud.

The background analysis clearly indicates that cloud computing is set to enjoy considerable further growth due to both the economic efficiencies that arise and the flexibility provided for software deployment. A review of the literature clearly suggests that the term "mobile cloud" is largely misrepresented, referring to smartphones leveraging the cloud as a thin client, rather than participating in the provision of cloud services. This is not surprising as in practice most mobile applications are statically partitioned to utilise the power of traditional cloud computing for back-end support. With the technical capabilities of smartphones increasing rapidly, the opportunity for mobile devices to play a role as service nodes at the edge of the Internet should be investigated. In certain circumstances, distributed computation by mobile devices at the edge may be more efficient given the literature highlights that significant energy is consumed in transmitting data when offloading computation.

While a mobile cloud could be used to emulate services that have previously been centralised, it is probable that a mobile cloud would play a different role and suit applications specifically designed to work with intermittent wireless networking and a highly distributed network of less powerful, battery constrained computing nodes. This raises the question as to whether existing architectures can be ported to mobile or whether an architecture designed from the ground up to accommodate mobile nodes would be superior.

Given the architecture of current datacenters, the unit cost of computation has decreased to the extent that a mobile cloud is unlikely to compete on grounds of pure processing efficiency, especially given the additional overhead of re-allocating processing tasks where mobile nodes have failed or left the network. However mobile nodes could analyse data collected from onboard sensors and reduce such data to an intermediary summary set for onward transmission.

An ideal implementation of a mobile cloud would accommodate heterogeneous smartphones in terms of both the hardware and operating systems (and versions of operating system). Middleware to support a mobile cloud would need to abstract users from this heterogeneity and provide a mechanism for managing a finely grained network of nodes that will experience node churn of a magnitude significantly higher than churn in a typical datacenter environment. It will not be possible in the medium term to run several virtual machine instances on a smartphone or to share one mobile broadband connection in a manner that would support several applications requiring always-on connectivity.

Peer-to-peer networking was analysed and provides a useful potential architecture for supporting a mobile cloud, given that nodes will naturally be at the edge of the network. Unstructured P2P networks, such as the Gnutella implementation, would incur too high of a message overhead in locating resources within a large mobile cloud. While the use of an unstructured P2P network coupled with a directory for locating resources is a viable option, it would require a well engineered distributed directory in order to be sufficiently fault tolerant. Structured P2P frameworks such as Pastry and Chord provide a robust manner in which to achieve a completely decentralised P2P network that can efficiently manage node churn and load balancing. However such networks, based on the use of distributed hash tables, require exact keyword searches and the 0(logN) hops required for file location may be better suited to fixed or ad-hoc wireless networks than to smartphones which will predominantly use 3G / UMTS and LTE. It would be inefficient from an energy perspective to require smartphones to

utilise wifi connectivity at all times. A key benefit though of structured P2P networks using DHTs such as Pastry is the fact that each node only needs to retain a subset of routing information rather than be aware of all nodes within the network.

The use of mobile Web services was considered. A mobile cloud built in part upon the use of Web services could allow a loosely coupled participation of heterogeneous mobile devices providing services. Given the nature of a lightweight web server this architecture would suit services that are accessed infrequently on mobile devices. But this option could support a personalized mobile cloud where anyone could participate in the mobile cloud and each participant would have complete control over access to and availability of their Web services. This suggests that mobile devices could be utilized to provide a personalised mobile cloud. In this scenario mobile nodes can be either federated to provide large computational resources, or rather utilized for the services each provides as a standalone mobile server. A key consideration for such architecture is the requirement for clear addressability and a supporting directory services such that personal Web services can be discovered. One benefit of mobile web services is that the device owner would be in complete control of the services offered and could simply turn on or off their personalized web services as they deemed fit.

In summary a mobile cloud raises some unique design challenges that requires an architecture tailored for a dynamic operating environment. A mobile cloud would lend itself to certain types of application such as disaster relief, decentralised social networks, proximity-based networks, smartphone sensor networks or general multimedia file sharing. In all cases the applications would need to be designed to operate within a mobile environment.

Having completed the evaluation of potential architecture that could be used, in part of whole, for the development of a mobile cloud, it is now important to choose a viable option for implementation. In almost all cases, with the exception of sensing applications, it is possible to construct a mobile cloud service model equivalent to those of a traditional cloud. With SaaS, PaaS, DaaS and IaaS the main drawback is that a mobile cloud would support only certain types of applications, in particular those that do not require heavy weight server nodes and stable or high-speed networking. The table below summarises the service models practically supported by a mobile cloud.

Table 7: Traditional versus mobile cloud service models

Cloud type	Traditional Cloud	Mobile Cloud	Comments
SaaS	Practically possible	Practically possible	Support certain applications.
DaaS	Practically possible	Practically possible	Mobile cloud limits file size.
PaaS	Practically possible	Not practical	No single platform provider.
IaaS	Practically possible	Not practical	Smartphones will not support VMs.
SeaS (Sensing as a Service)	Not practical	Practically possible	Possible to access smartphone sensors.

While it is possible to develop a mobile processing cloud, this option is ruled out given it is not as practical as other potential applications. Given the benefits associated with a mobile storage cloud the next chapter of this dissertation will focus on implementing such a system.

5. IMPLEMENTATION & EVALUATION

The literature analysis pertaining to cloud computing, and particularly how mobile nodes might participate, highlighted that there are a variety of options for implementing a mobile cloud. These options include a mobile processing cloud, provision of personalised Web services or a mobile storage cloud. The optimal option depends on the deliverables of the system and the architecture is therefore dependent on the requirements specification of any system. Given the medium term energy constraints, the limited processor capabilities and the messaging overhead that arises, the option of a mobile processing cloud has been ruled out. While the provision of personalised Web services on mobile devices appears to have significant potential, this option was also ruled out in favour of implementing a mobile storage cloud. Such a system could support a wide variety of applications including the dissemination of critical information in a disaster zone, a mobile distributed social network, or the sharing of sensor data over a large geographic area. The system, hereon referred to as Icarus, is designed to be fault tolerant against mobile nodes churning in a network.

The implementation assumes that all mobile nodes can be directly addressed. This is a minimum requirement to implement any of the options considered. This is not currently the case, as highlighted earlier, but the transition from IPv4 to IPv6, will resolve addressability in the medium term and is currently underway. The assumption that mobile nodes are addressable is therefore reasonable in the context of this study.

5.1. Requirements specification

This section will provide an outline of the requirements specification for Icarus, which will be used to guide design decisions. Given the opportunities that exist to enhance and extend the system it has been critical from the outset to clearly highlight the deliverables that are inside of scope and those that provide opportunity for future extension. The following is a list of the requirements included in the prototype.

1. P2P file retrieval. Once an Icarus node locates the required information and responds to a remote request, it will return sufficient information to allow the requesting Icarus node to directly contact the node hosting the required content. This will facilitate direct P2P content retrieval.

2. Join / Leave: The system should be able to tolerate Icarus nodes joining and leaving the network. For the prototype the ability to join will be semi-automated, in that a joining node will be required to know the IP address of at least one other Icarus node.

3. Fault tolerance: Any participating mobile node may experience hardware failure, intermittent network connectivity or a loss of battery power. A network of Icarus nodes is expected to be able to outlive a reasonable level of node departure or failure.

4. Search: Unlike DHT based P2P systems, Icarus should allow for inexact searches. The system should ensure that the number of messages hops required to locate content can be minimised in order to alleviate potential message failure due to network congestion or intermittent availability.

5. Practicality: The system is designed to support distributed storage for the sharing of information that can be transmitted across a wireless network efficiently. Icarus is not designed for the transmission of large files such as video. The system could be augmented in the future to allow for file fragmentation across multiple nodes.

6. Energy efficiency: As smartphones are battery operated and wireless transmission consumes significant energy, an Icarus network should support fair load balancing across nodes. This should be achieved by randomly choosing an IP address from the pool of available mobile nodes to service requests.

7. Scalability: The network should be able to scale. An Icarus node should not need to be aware of all other nodes within the network.

8. Replication: The objective is not to ensure the availability of each file stored on an Icarus node, but rather to ensure the availability of at least one copy of the file in the system irrespective of reasonable levels of node churn or network failure.

9. Language: The prototype will be implemented in Java.

A prototype Icarus node has been developed in Java. Each node has a client, directory and storage node component. The directory component has the capability to accept inbound queries and to transmit requests from, and to, the directories of other Icarus nodes. The diagram below illustrates this network topology. Once a client submits a request to put a file into, or retrieve a file from the network, it communicates to its local directory. There are five steps required in adding content to an Icarus network:

1. The user accesses the Icarus client.

2. Local Icarus node checks to ensure the file exists (functionality in local directory).

3. Local Icarus node randomly chooses a neighbour peer node from its routing table and requests that it replicates the content.

4. The neighbour node chosen requests the file from the Icarus node, using a unique URL for that content.

5. The local directory in parallel replicates the metadata associated with that content and transmits it to the random directory chosen. A copy of metadata will exist for each replica, as each will have a unique URL. Metadata can be propagated to other directories to increase the speed at which any replica can be located within the network.

5.2. Client design

Limited focus has been placed on the client interface design for the prototype. It is designed merely to allow for the running of simulations to evaluate the performance of Icarus. It is envisaged that the user interface could be developed to allow Icarus to be deployed as an Android application using MySQL lite. The diagram below provides an illustration of the current menu screens. Within the main menu, option 5 provides access to the simulation submenu.

Diagram 12: Icarus – user interface

5.3. Directory design

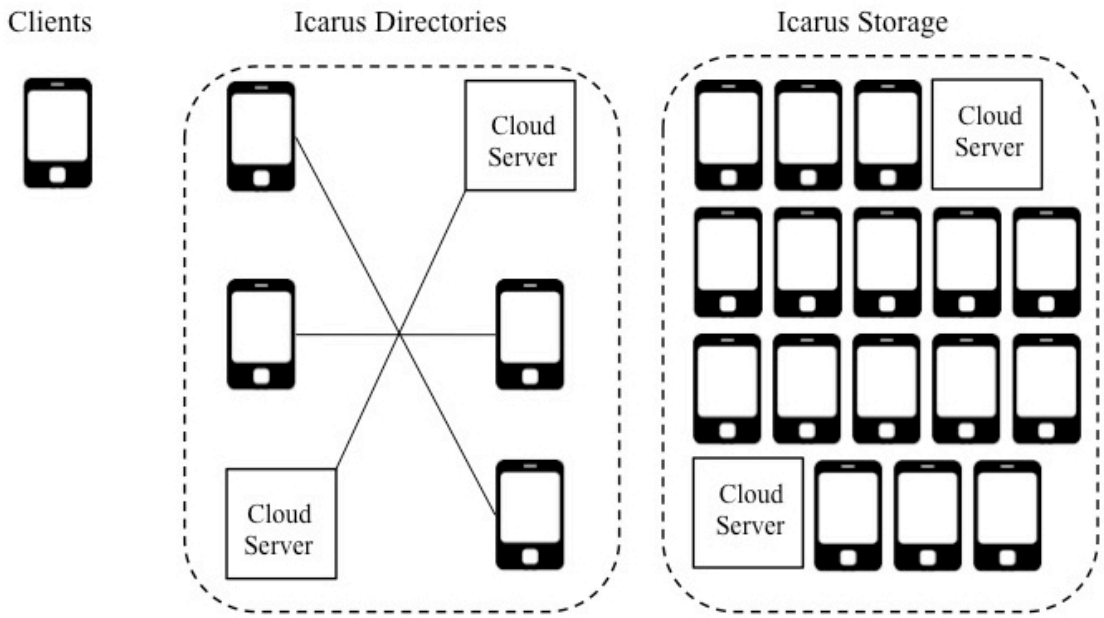
The design of the Icarus directory is a fundamental component of the system. Each Icarus network consists of one overall directory, with this directory being distributed and replicated across all of the participating nodes within the network. Each node is therefore responsible for maintaining a portion of the overall directory.

5.3.1. Peer neighbourhood

It is an objective of the design to ensure that each Icarus node does not need to be aware of all other nodes participating within the network. Rather any node must only maintain a limited routing table incorporating a small subset of peers, referred to as neighbour peers. These neighbour peers do not need to be within close physical proximity to the node. The number of neighbour peers listed within the routing table is a parameter that can be tuned within the network. The evaluation is based on this parameter being set at 10. Upon a request, if an Icarus node needs to contact another node to pass on a request, it chooses one of these nodes at random from its routing table. The node contacted from the neighbour list may in turn route the message to one of its own neighbours, also chosen at random. The key decision criteria in setting the neighbourhood parameter is to ensure that at least one of the neighbour peers will be available at any time to service a remote request. A neighbourhood set of 10 nodes would ensure availability of the network so long as all 10 nodes are not unavailable simultaneously. A real life deployment of the network should ensure the random nodes chosen for the neighbourhood set reside across different wireless networks when possible. This will further improve the fault tolerance of the network, limiting the potential for multiple nodes to be adversely impacted by a core network failure. A while loop is used to randomly choose a node from the neighbourhood set in order to query a remote directory for a file. If the node chosen is unavailable, the requesting node will choose another random neighbour peer.

The diagram below illustrates the network topology for Icarus.

Diagram 13: Icarus – components



An Icarus node can also reside on a traditional datacenter server, thereby allowing for super nodes that could incorporate complete copies of the entire network directory data or larger partitions of this data relative to the participating mobile nodes. Such an implementation requires that the address of such super nodes be given priority within the peer neighbour routing table. Even in the event that a super node failed, such failure is tolerated and requests will default to the remaining distributed nodes and be distributed on a random basis. Such a deployment also ensures that at least one of the directory peers used is on a separate physical network thereby improving fault tolerance. The benefit of such an implementation is that the node hops required to locate content could be reduced to as little as 1.

5.3.1. Content and metadata replication

The replication of content and directory data is imperative to ensuring availability and performance within an Icarus network. For each replica of content, the metadata including a

URL pointer to the content is also replicated separately to the directory partitions of other random Icarus nodes. This ensures that the overall directory is robust and the failure of one or more Icarus nodes should not result in an inability to locate content in the storage network. There are a number of options available with regard to the replication of the content itself. A key design choice is whether the replication decisions are made by the user of the system or by the Icarus nodes' local directory, in which case the replication policy is transparent to the user. Two options for the replication of content and metadata were considered:

1. Directory logic decides on how to replicate the file. This could be based on the file type and a timestamp. The use of a timestamp indicating when content was added to the network could allow for an automated purging of replicas after a period of time to optimise storage resources across the Icarus nodes.

2. User decides on how to replicate the file. The client could provide an option allowing for a user to choose a replication policy based on a range of predefined options. For example, the client could ask the user to weight the importance of the file availability on a scale of 1 to 5, where 1 reflects a very important file such as medical data and 5 represents a file of lesser importance.

Two further options arose with respect to replicating metadata within the distributed directory.

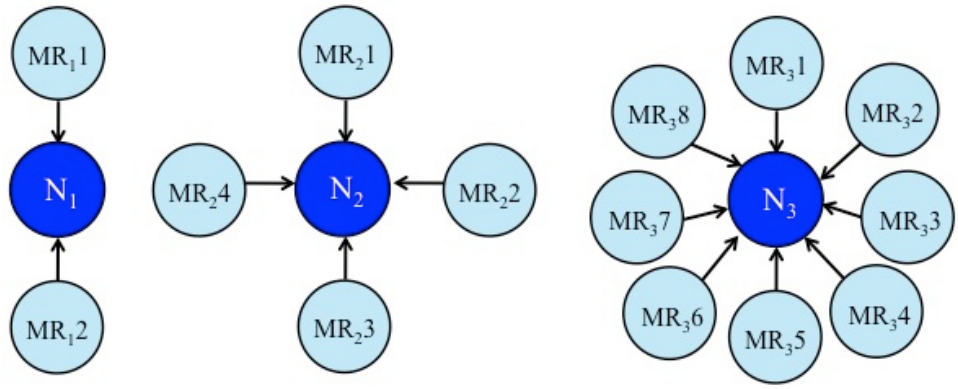
1. Replicate directory partitions across nodes such that the partition of a directory hosted on a node is exactly mirrored on at least one other node within a network.

2. Replicate content metadata on a file-by-file basis and propagate this to the directory data of multiple Icarus nodes chosen randomly.

The latter option was chosen as it supports the prioritisation of different content. With this method the metadata for content can be replicated dynamically based on its importance. The metadata for high priority content is propagated to the directory partitions of a greater number of Icarus nodes, thereby requiring fewer node hops in order for a search request to locate the

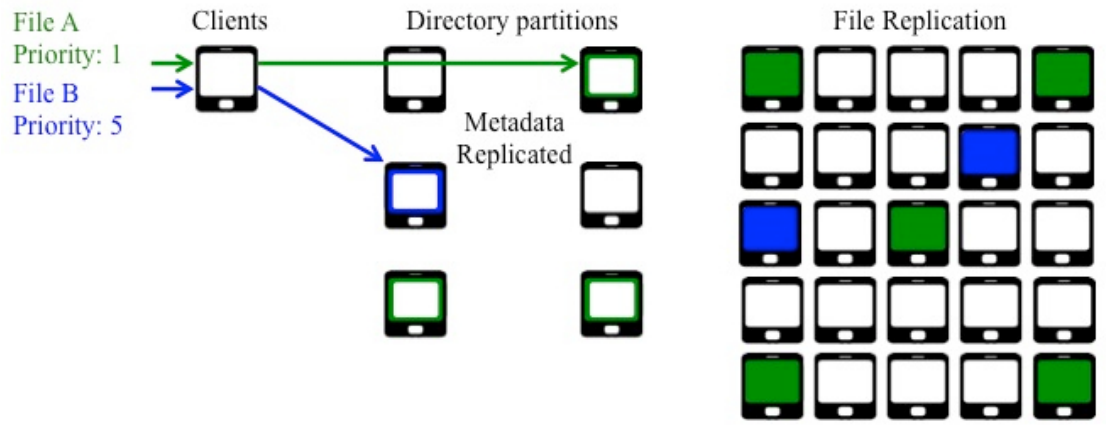
content. This concept of dynamic replication is illustrated in the diagram below. In this instance the directory data and URL pointing to a file on Icarus node 1 (N_1) has been replicated twice. In the case of Icarus node 3 (N_3), the metadata has been replicated eight times. In this case the metadata will therefore reside within the directories of eight separate Icarus nodes.

Diagram 14: Dynamic replication



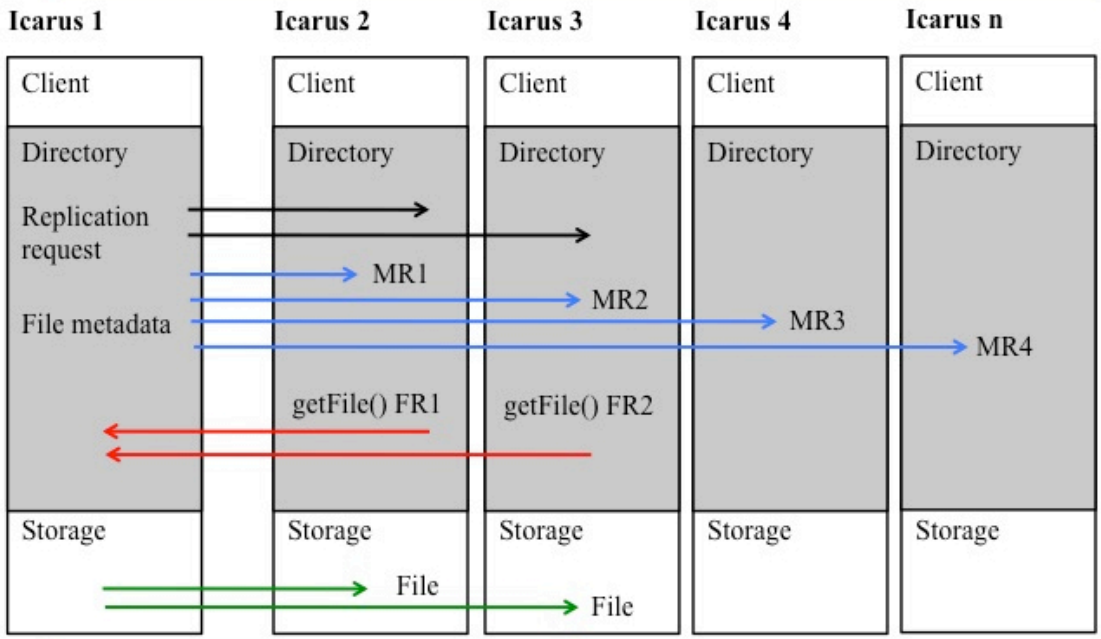
The benefit of this architecture is that content of high importance can be more quickly located within the network, given that more Icarus nodes are aware of its existence and location. The diagram below provides a clear illustration of the combined implications of replicating content and metadata separately. In this case file A represents a high priority file, whose metadata is propagated to three directories, with file replicas of the content itself hosted in the storage component of separate Icarus nodes.

Diagram 15: Replication illustrated



The actual operation of adding a file to the network is provided in more granular format below. Here Icarus node 1 is adding a file to the network. This results in a replication request being submitted to nodes 2 and 3. Both of these nodes then execute the getFile method in order to retrieve a copy of the file from node 1. The metadata and URL for the file are replicated to nodes 3 and 4. While these nodes do not contain a physical copy of the file itself, they can advise any other nodes within the Icarus network of the location of the file.

Diagram 16: Directory replication request



This concept of dynamic replication ensures that valuable storage resources and network messages are not utilised inefficiently by treating all content as equal. As a result the resources available across an Icarus network are utilised more efficiently. The availability of priority content, even assuming significant node churn, can be tuned both to maximise fault tolerance and to ensure the content can be located quickly in the network. This mechanism allows for content categories, that could range from 1 to 5 to be implemented whereby the category becomes a field value in the metadata associated with a file, highlighting the replication policy that should apply. The replication algorithms do not take account of the size of the Icarus network given that each node is not actually aware of how large the network it is participating within is. To incorporate this functionality would result in a significant messaging overhead across the network and this overhead is not considered a required pre-requisite to deploying a reliable mobile storage cloud.

The architecture results in the number of replicas within an Icarus network being inversely related to the number of node hops required to locate content within the overall Icarus directory. By further replicating only the metadata across nodes, the performance and availability of the network can be further optimised as content can be more quickly located across fewer node hops. The availability within an Icarus network is calculated as 1 – P(all replicas failing) where the probability of all replicas failing is determined by a hypergeometric distribution. The possible combinations (Combin) of nodes failing within an Icarus network can be denoted as:

(Combin) = N!/c!(n-c)! where:

N = total nodes in the network.

c = total nodes churning.

To determine the probability of whether some or all of the file replicas required are amongst the nodes that fail the following formula is used. This is denoted as:

P(replicas failing) = (Combin(r, f) x Combin(N-r, c-f)) / Combin(N, c) where:

r = number of file replicas

f = number of replicas failing

Therefore a simple Icarus network with 20 nodes and 10% node churn, would have the following number of potential combinations of nodes churning:

= 20!/2!(20-2)!

= 2,432,902,008,176,640,000 / 2 / 6,402,373,705,728,000

= 190 combinations.

The table below extends the above analysis to highlight the combinations of nodes that could churn in a network of 50, 75 and 100 nodes.

Table 8: Combinations of nodes churning

Network nodes (N)	50	75	100
P(first five failed all had replicas)	0.0000472%	0.0000058%	0.0000013%
P(1st node fails has replica)	10.00%	6.67%	5.00%
P(2nd node fails has replica)	8.16%	5.41%	4.04%
P(3rd node fails has replica)	6.25%	4.11%	3.06%
P(4th node fails has replica)	4.26%	2.78%	2.06%
P(fifth node fails has replica)	2.17%	1.41%	1.04%

Formula for combinations (Combin)	N!/c!(N-c)!
Formula P(all replicas fail)	P(all replicas fail) = (Combin(r, f) x Combin(N-r, c-f)) / Combin(N, c)

Network nodes (N)	50	75	100
Churn rate C	10%	10%	10%
Nodes churning (c)	5	7.5	10
File replicas (r)	5	5	5
Replicas failing (f)	5	5	5
N!	3.04141E+64	2.4809E+109	9.3326E+157
c!	120	5,040	3,628,800
(N-c)!	1.19622E+56	3.64711E+94	1.4857E+138
Combinations nodes failing	2,118,760	134,968,429,800	17,310,309,456,440
Probability of replicas failing	0.0000472%	0.0001217%	0.0003347%
File availability	100.0000%	99.9999%	99.9997%

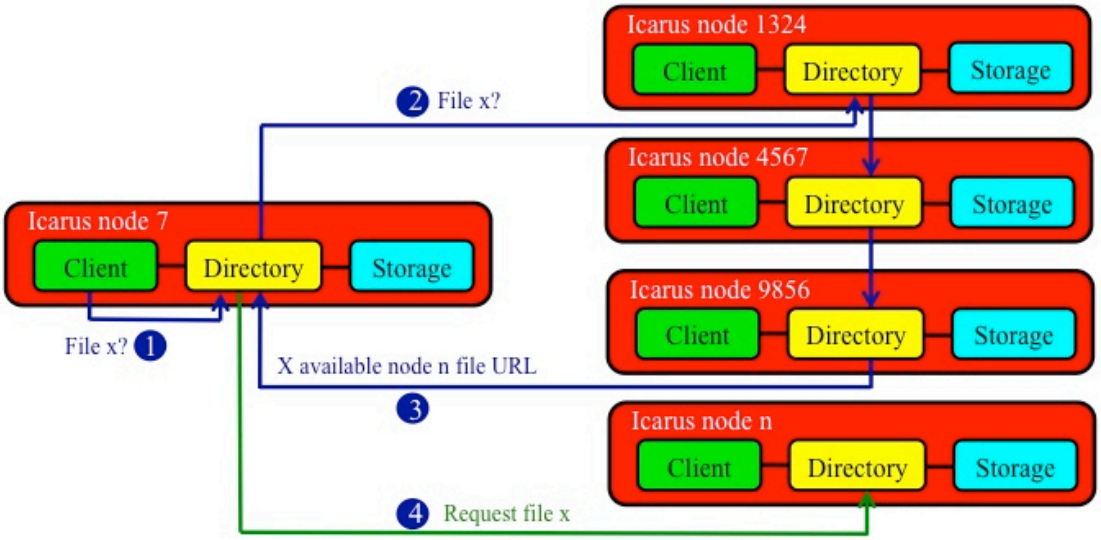
In a network with 100 Icarus nodes and 10% churn, it would be possible to have 17.31 billion combinations of the 10 nodes that churn in the network calculated as100!/10!(100-10)! The probability of 5 nodes hosting replicas of content failing within a network of 100, where 10 nodes fail overall should be 0.0003347%. Availability should therefore be 1-0.0003347% or 99.9997%. The above analysis is based on the probability of a node failing not being directly correlated to any other node failing. This may not always be the case given a physical failure within the core of a wireless network could result in multiple mobile devices failing. As a result the random nodes chosen for the neighbour peer routing table and for hosting replicas and metadata should be as diverse as possible and span multiple physical networks if available.

5.3.2. Search and content retrieval

In order to search Icarus a client submits a search string. This search string is translated into a query to the node's local directory. This checks for a full or partial match in its database and responds if a match is made. If a match is not made, then the local directory submits the original string query to another random directory on an Icarus node in its neighbourhood routing table. Once the file is located by one of the directory partitions, the URL for that file is returned to the originating node requesting the file. This URL consists of the IP address of the node, the port to request the file on, a hash of the file and the filename. The client is provided with the results from a search query with the associated URLs of any files found. The client can then submit a URL (in the form http:\\134:226:34:10000/hash_file/filename to directly retrieve the file from the node that it resides on. The diagram below illustrates this search management logic.

While the input of content to Icarus or a search query for content may involve multiple hops in the Icarus network, the content retrieval is based on a direct P2P connection. This is illustrated in the diagram below and ensures efficient content retrieval.

Diagram 17: Search and content retrieval



5.4. Storage design

The storage component of Icarus is relatively simple. It is responsible for receiving inbound requests for content and either servicing that request or indicating that the content is not available. Two options existed:

1. Use Java sockets to receive content requests and respond with the requested content.

2. Use RESTful Web services in order to engineer a more loosely coupled system. This option would be of benefit given it would more easily support the participation of heterogeneous mobile nodes.

The use of Web services requires that a cut down version of the Apache web server or equivalent be deployed on each Icarus node. Each file within an Icarus network would then have its own unique URL that could be accessed on any storage node. For the prototype Java sockets were used for the transmission of the file but unique URLs are also used to locate Icarus node where the content resides and the file path for the file on the node.

5.5. Database schema

Data is persisted on each Icarus node using a database. For the prototype network, each node had MySQL installed. The system will however support SQLite (installed by default on all Android based handsets) or a text based SQL flat file database such as TextDB. The database consists of 5 tables including: files, meta, replicaURL, searchrequests and searchresults. The layout of each of these tables is provided below.

```
mysql> show tables;
+------------------------+
| Tables_in_icarussim999 |
+------------------------+
| files                  |
| meta                   |
| remotedir              |
| replicaURL             |
| searchrequests         |
| searchresults          |
+------------------------+
6 rows in set (0.00 sec)
```

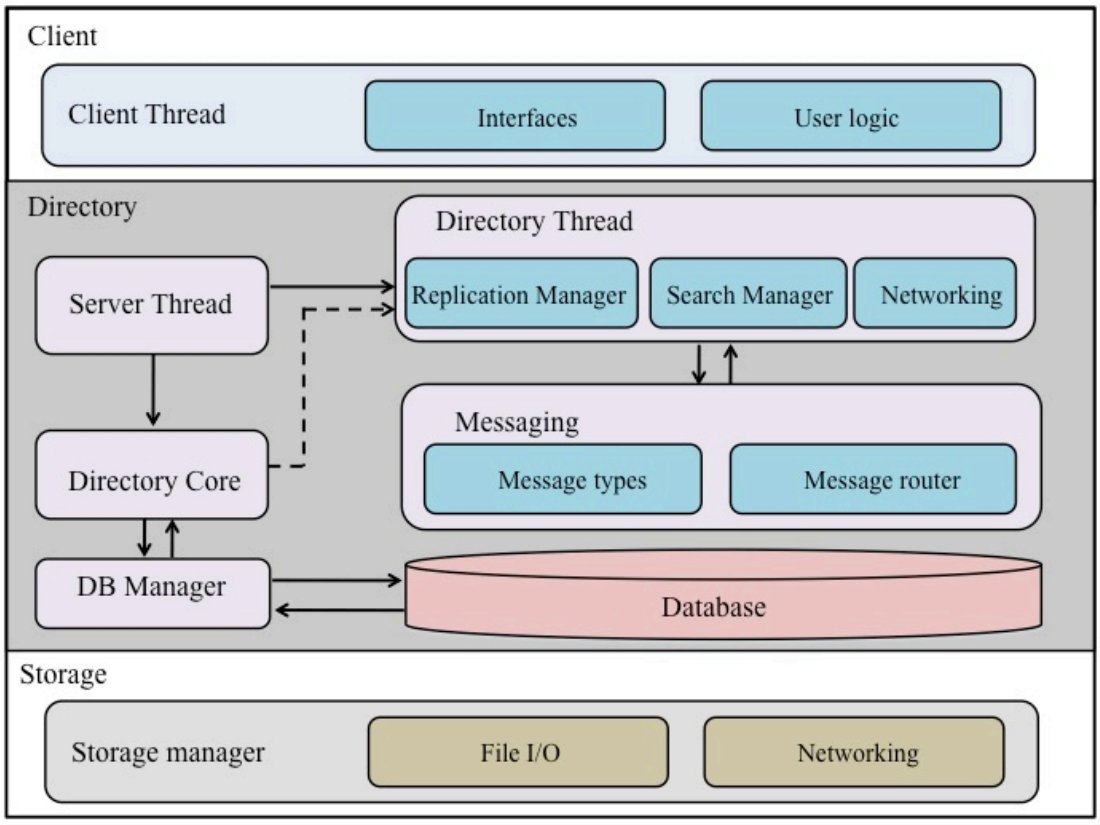
```
mysql> show columns from files;
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| fid           | int(11)       | NO   | PRI | NULL    | auto_increment |
| hash          | varchar(40)   | YES  |     | NULL    |                |
| local         | tinyint(1)    | YES  |     | NULL    |                |
| ttl           | int(11)       | YES  |     | NULL    |                |
| localfilepath | varchar(2083) | YES  |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> show columns from meta;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| fid      | int(11)       | YES  |     | NULL    |       |
| metaid   | int(11)       | YES  |     | NULL    |       |
| metaText | varchar(2083) | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> show columns from remotedir;
+-------+---------------+------+-----+---------+----------------+
| Field | Type          | Null | Key | Default | Extra          |
+-------+---------------+------+-----+---------+----------------+
| did   | int(11)       | NO   | PRI | NULL    | auto_increment |
| host  | varchar(1000) | YES  |     | NULL    |                |
| port  | int(11)       | YES  |     | NULL    |                |
+-------+---------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> show columns from replicaURL;
+------------+---------------+------+-----+---------+----------------+
| Field      | Type          | Null | Key | Default | Extra          |
+------------+---------------+------+-----+---------+----------------+
| fid        | int(11)       | YES  |     | NULL    |                |
| rurlid     | int(11)       | NO   | PRI | NULL    | auto_increment |
| replicaURL | varchar(2083) | YES  |     | NULL    |                |
+------------+---------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)

mysql> show columns from searchrequests;
+----------------+---------------+------+-----+---------+----------------+
| Field          | Type          | Null | Key | Default | Extra          |
+----------------+---------------+------+-----+---------+----------------+
| searchstringid | int(11)       | NO   | PRI | NULL    | auto_increment |
| searchstring   | varchar(2083) | YES  |     | NULL    |                |
+----------------+---------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

```
mysql> show columns from searchresults;
+----------------+---------------+------+-----+---------+----------------+
| Field          | Type          | Null | Key | Default | Extra          |
+----------------+---------------+------+-----+---------+----------------+
| searchresultid | int(11)       | NO   | PRI | NULL    | auto_increment |
| searchstringid | int(11)       | YES  |     | NULL    |                |
| resultlocation | varchar(2083) | YES  |     | NULL    |                |
+----------------+---------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

5.6. Summary architecture

A high level overview of the Icarus architecture is illustrated below. The diagram highlights the client, directory and storage functionality that will be incorporated into each Icarus node. The main method resides within the directory server class and also instantiates simple client access from where a user can navigate the system.

Diagram 18: Icarus - architecture



The directory is multithreaded and spawns a thread to manage each request to a node's directory. This allows for the concurrent management of requests. The Directory Core is a shared directory object that manages the addition of local files to Icarus, adding a remote file as local (upon receipt) and querying the storage node with the hash of a file. The Directory Core also manages setting up the peer neighbour routing table.

In summary each Icarus node consists of a client, directory and storage component. The directory component incorporates the intelligence to internetwork remote directories, handle file and metadata replication and manage search requests. The storage component receives content requests and responds with that content.

5.7. Evaluation

The evaluation consisted of deploying an Icarus network across several mobile devices and datacenter virtual machines, followed by the development of a simulation package to allow extensive testing of larger networks. The evaluation focused on the following key objectives:

1. Testing network availability in the presence of progressively worsening levels of node churn.

2. Logging and evaluating the number of node hops required to locate content for a search request. This entailed adding counter functionality that is incremented each time a message is transmitted across nodes.

3. Trial experiments to put content into the network, replicate the content and metadata separately across nodes and subsequently search and retrieve the content.

4. Stepwise iteration of the experiments to determine the ability to tune the network for availability and performance by varying the number of file and metadata replicas.

A number of simulation options were considered to evaluate an Icarus network. These included JiST / SWANS, TOSSIM and NS2. JiST / SWANS refers to Java in Simulation Time / Scalable Wireless Ad Hoc Network Simulator, a high performance network simulator that runs on a standard Java virtual machine. JiST simulations are written in Java. SWANS provides a scalable wireless network simulator built on top of the JiST platform. It leverages JiST to run standard Java network applications over simulated networks. TOSSIM and NS2 provide scalable simulators for simulating wireless sensor networks (WSN). Of the three simulators JiST / SWANs provided the closest match to the requirements to test an Icarus network, given it allowed for Java sockets to be translated into emulated network sockets. These simulators provided some advanced capabilities, such as the ability to simulate radio

interference within a wireless network. However, the simulators focused on ad hoc networking rather than simulating performance across a wide area 3G or LTE wireless network. While an Icarus network could be deployed across multiple WiFi networks, it should be evaluated based on the levels of node churn that may be expected in a standard wireless network. As none of the simulators provided the ability to test Icarus comprehensively across the four evaluation criteria, a separate simulation package was developed.

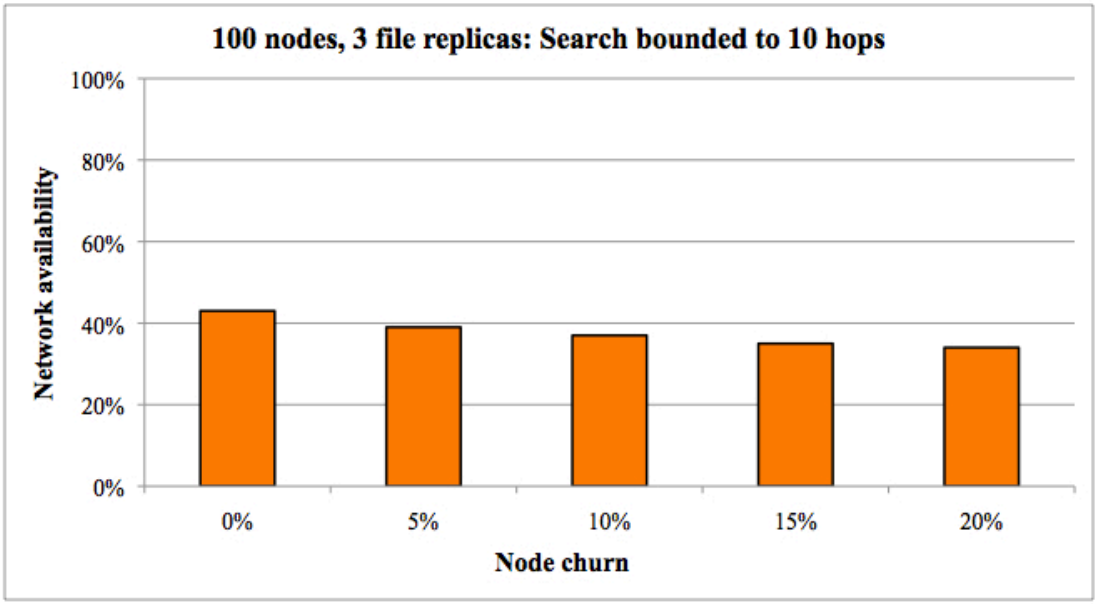
The simulation package allows for key parameters to be flexed including the following:

1. Number of neighbour peers. The default setting was 10.
2. How many Icarus nodes are in the simulated network.
3. The volume of iterations to complete for each experiment.
4. Churn rate applying to any simulation.

For all experiments, a minimum of 100 iterations is completed in order to ensure data that is representative. The simulator spawns a thread for each Icarus node within the network. It creates the underlying directory database tables required for each node and empties the data upon completion of an experiment.

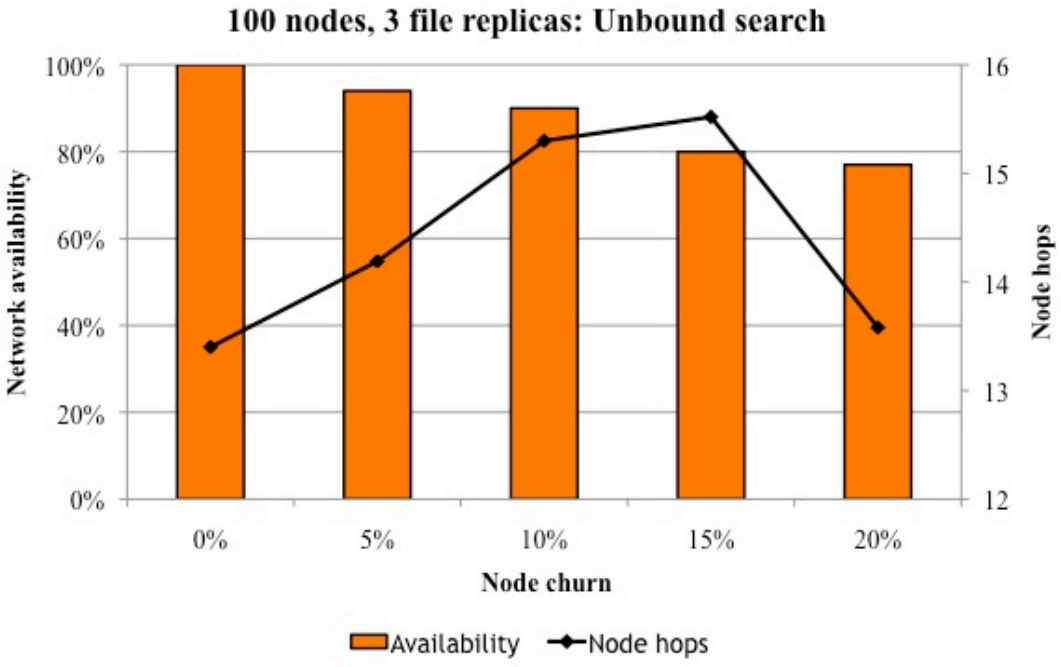
The graph below illustrates the outcome of a simulation using 100 Icarus nodes with 3 copies of a file randomly stored within the network. For each simulation 100 iterations of storing a file and subsequently searching for and retrieving the file are completed. The simulations are repeated successively increasing the churn in increments of 5%. The total number of experiment iterations completed is therefore 500 (100 x 0% / 5% / 10% / 15% / 20% churn). The number of hops that a search query can traverse is limited to 10. This results in a network availability that ranges from 34% to 43% depending on the churn rate. The poor availability is caused by the search constraint of 10 node hops. The 3 replicas of the content are often not located within the network using 10 random search hops.

Graph 2: Availability – search bounded to 10 node hops



If this search hop constraint is removed, the ability to locate the content shows a clear improvement. However this improvement in availability incurs the overhead of the additional node hops as illustrated below.

Graph 3: Availability – unbounded



The increased node hops adversely impacts the latency associated with responding to a request for content within the network. It also increases the probability that a request will fail.

The following four scenarios were then simulated. In each case, 100 iterations of the experiment to put content into an Icarus network and subsequently search and retrieve the content was completed.

Scenario 1. Network with 100 nodes, 3 file replicas (FR), metadata replicated to 0 nodes.

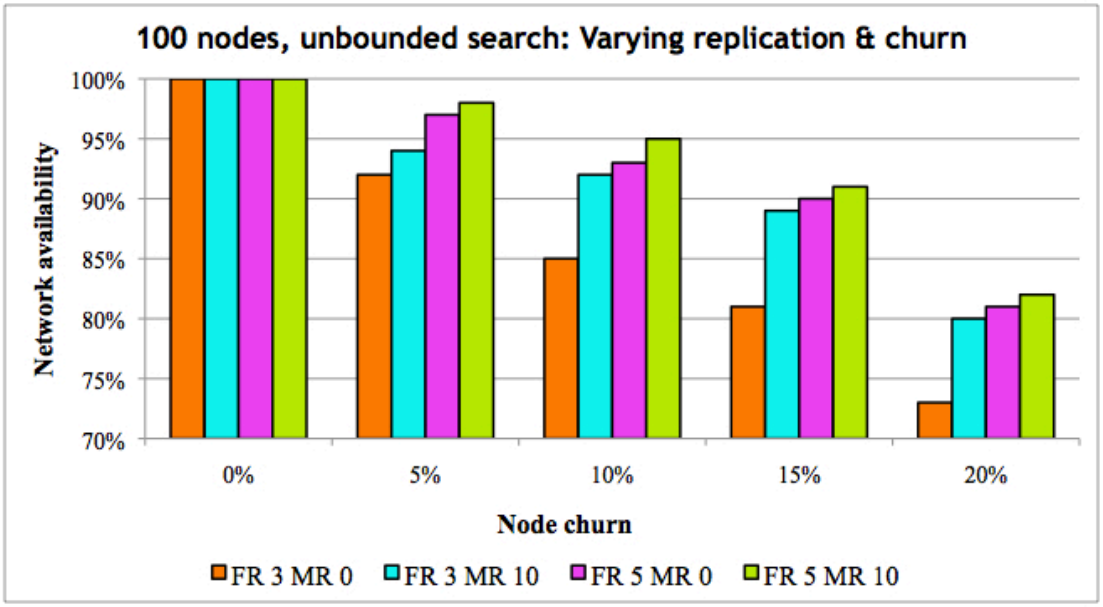
Scenario 2. Network with 100 nodes, 3 file replicas (FR), metadata replicated to 10 nodes.

Scenario 3. Network with 100 nodes, 5 file replicas (FR), metadata replicated to 0 nodes.

Scenario 4. Network with 100 nodes, 5 file replicas (FR), metadata replicated to 10 nodes.

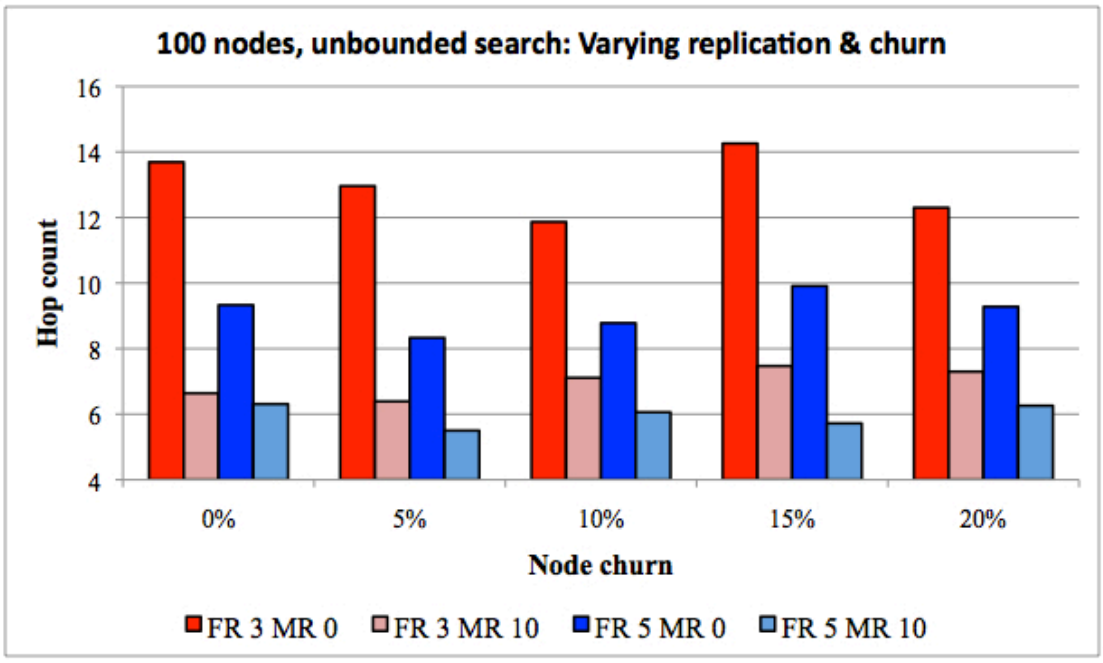
These experiments involved running the simulations for several days. For each scenario the churn was varied from 0% to 20% in 5% increments. For each scenario 500 experiment iterations was completed (100 iterations x 0% / 5% / 10% / 15% / 20% levels of churn), resulting in a total of 2,000 experiment iterations. Each of these iterations involved randomly replicating the content and metadata placed in the network across nodes. The graph below illustrates the availability of Icarus networks. It becomes clear that separately replicating the location metadata across Icarus directories increases the availability within an Icarus network. This is because there are no more directory nodes that are aware of the location of at least one of the file replicas. The improved availability resulting from the replication of location metadata is greater at higher levels of node churn. The availability of a network with 3 replicas of content and with the location metadata replicated to 10 directory partitions approaches that of a network with 5 replicas of the content and no metadata replication. This finding is useful given that the cost of replicating metadata will be significantly lower in storage terms than for replicating actual content.

Graph 4: Availability levels at varying replication



The graph also demonstrated the ability to tune availability within the network to in excess of 95%, despite a significant number of nodes churning off the network simultaneously. The graph below illustrates that the replication of metadata across Icarus nodes considerably impacts the number of node hops required to locate content within the network.

Graph 5: Network search – hop count to locate content



On average content is located using half as many node hops, vastly improving the latency in responding to a client request and reducing the potential for request failure due to message loss. While Icarus was not designed to compete directly against distributed hash table based networks, it can be tuned to outperform a Pastry network on search by increasing the metadata replication such that the number of hops required to locate content is less than Pastry's 0(log N). As outlined previously, the number of node hops can also be reduced to one through the use of super nodes hosting an entire directory.

6. CONCLUSIONS

The literature misuses the term “mobile cloud” by referring predominantly to mobile devices that access and leverage a traditional cloud as a thin client. Given the exponential growth in the use of smartphones globally and the increasingly rich computation resources they offer, it is worth investigating the ability to federate such devices into a mobile cloud. Afterall, Metcalfe’s law suggests that the value of any network increases by the square of the number of nodes.

The background analysis indicates several developments that fostered the emergence of cloud computing. During the 1960s virtualisation was developed for allocating mainframe resources, but later waned due to the commoditisation of hardware, only to re-emerge for allocating computing resources within datacenters. Improvements throughout the 1980s and 1990s reduced commodity hardware and networking costs, stimulating migration to horizontally scalable datacenters. Finally, automating resource provisioning via online portals enabled the flexibility and efficiency of federated computing to become widely accessible.

Evaluating the implications of mobile server nodes highlights numerous constraints for implementing a mobile cloud. From a hardware perspective mobile nodes have less processing power than commodity servers and are battery constrained. Smartphones were never designed to be used as always-on serving infrastructure. The heterogeneity of mobile hardware and operating systems is also a concern. Given these differences, a mobile cloud will exhibit different failure semantics than a traditional cloud, especially with regard to availability and performance. Wireless networks suffer from lower bandwidth and greater intermittency, resulting in a degradation of the performance of transmission protocols. TCP

overreacts to temporal congestion, and its large packet headers and three-way handshake would be a high overhead when transmitting maintenance messages within a mobile cloud.

The analysis explored the potential implementation options for a mobile cloud. Given the warehouse-scale architecture of current datacenters, the unit cost of computation has decreased to the extent that a mobile cloud is unlikely to compete on the grounds of pure processing efficiency. This is due in part to the additional overhead of re-allocating processing tasks where mobile nodes have failed or departed the network and to the greater messaging overhead in wireless networks. While prior research has been completed on mobile distributed processing, this option was ruled out due to the constraints outlined above. However, in certain circumstances, distributed computation by mobile devices at the edge of a network may be efficient, as significant energy is consumed in transmitting data when offloading computation. Mobile nodes could, for example, analyse data collected from onboard sensors, analyse the data, and transmit summary data to super nodes.

The potential of peer-to-peer networking was considered and provided a useful architecture for supporting a mobile cloud. While structured P2P networks such as Pastry and Chord are well engineered and use consistent hashing to provide a defined address space, they require exact keyword searches for file retrieval. Though the ability to locate data within a P2P network based on 0(logN) hops may be considered efficient in a fixed network, it may not be optimal in a wireless network characterised by low and erratic bandwidth, as the probability of message failure increases with each additional node hop.

The use of mobile Web services was explored, which would allow for a loosely coupled cloud design whereby heterogeneous mobile nodes could participate in providing services. Given the general constraints that arise with mobile nodes, a lightweight web server is required that could host services that are accessed infrequently on mobile nodes. This option could support a personalised mobile cloud where anyone could participate and with each participant having

complete control over access to, and availability of their Web services. In this scenario mobile nodes could be either federated to provide large computational resources, or rather utilised for the services each provides on a standalone basis. Such architecture would require clear addressability and a supporting directory enabling personal Web services to be discovered. The ability for a provider of mobile Web services to have physical control of the services hosted may be attractive for privacy reasons, as personal data in an application like mobile social networking would not have to reside on remote servers.

Distributed file systems are not currently optimised for a mobile environment. Rather, they are designed for an environment where nodes and network connectivity are reasonably stable. The topology is usually based on centralised servers with fault tolerance provided by redundant hardware and the static replication of data. A mobile storage cloud will require a different architecture from a traditional cloud, given the participation of mobile server nodes raises unique problems that would not arise in a traditional cloud. A network of mobile nodes will dynamically change in real-time as nodes constantly join and leave the network. These differences result in a requirement to define failure semantics for a mobile storage cloud that are different from a traditional cloud, where each commodity server may have an annualised availability rate of 99% or higher. The users of a mobile storage cloud need to be abstracted from the instability of a network in which the underlying service nodes are dynamically changing. This requires a solution to two unique problems posed by a mobile storage cloud. First, the availability of the network needs to be architected from the ground up to ensure a reasonable level of service can be provided irrespective of a proportion of the network nodes changing at any given time. Second, given that a mobile storage cloud is highly distributed, the ability to search nodes and to locate and retrieve content needs to be optimised in order to reduce the messaging within the network and minimise the associated latency in responding to search requests.

Icarus incorporates many of the lessons from the analysis. Its peer neighbourhood design ensures that nodes need only be aware of a small subset of nodes within an overall network. This parameter can be flexed when deploying a specific network environment to guarantee a high probability that at least one peer node will be available upon request. The neighbourhood design helps ensure that the network maintenance messages transmitted are minimised, with no requirement to flood an entire Icarus network to determine the global state at any given time. To overcome the availability challenge, content is replicated to a group of mobile nodes. This provides an innovative architecture where Icarus' fault tolerance is based on the conditional probability that all of the mobile nodes containing the requested content are unavailable or fail simultaneously. Such a conditional probability is analogous to drawing all of the numbers within a lottery draw. Dispersing the content replicas across Icarus nodes on multiple physical networks when possible removes the potential for a single point of failure to trigger a cataclysmic failure.

The separate replication of metadata and URL significantly decreases the messaging cost and latency in locating data. Given TCP's poor performance in wireless networks, the reduced message hops has a large impact in ensuring requests don't fail during transmission, as fewer peer connections are required. The Icarus architecture allows for super nodes to be implemented as preferred peer neighbours that could host an entire directory. This would reduce the directory hop count to one with the ability to default to the Icarus directory partitions on each smartphone if a super node fails. The replication of content and metadata can be applied dynamically, depending on content importance, allowing for the efficient allocation of storage and directory resources within the network.

A mobile storage cloud could support a suite of next generation applications including disseminating information in a disaster zone where traditional resources are unavailable, providing mobile distributed social networking, collating data from distributed mobile sensor applications or simply sharing multimedia content.

6.1. Future work

The analysis also highlighted several areas for future research. These include:

1. Enhancing and extending Icarus. Appendix 1 outlines areas for improving Icarus.

2. Dynamically partitioning mobile applications between local mobile on-device computation and datacenter computation to optimise the use of available resources.

3. Research into developing a thin hypervisor for mobile devices that could support the efficient operation of isolated mobile virtual machines.

4. Mobile grid. Further research is warranted on the potential to develop a distributed mobile processing grid. Such a grid could be used for a variety of novel applications including image analysis.

5. Delay tolerant network. Given the poor performance of TCP in wireless networks it may be worth researching the potential to use delay tolerant networking. Asynchronous notifications and transmissions could help alleviate issues that arise with TCP.

7. BIBLIOGRAPHY & APPENDIX

7.1. Bibliography

[1] *Amazon Web Services.* http://aws.amazon.com/ (accessed 2011 йил 17-April).

[2] Anderson, Tom, et al. “A Case for Networks of Workstations.” *University of California, Berkeley*, 1992.

[3] *Android x86 Project.* 2010. http://www.android_x86.org.

[4] *Apache Open Source Web Server.* http://www.apache.org (accessed 2011 йил 2-August).

[5] Barham, P., et al. “Xen and the art of Virtualization.” *Nineteenth ACM symposium on Operating systems principles* (ACM), 2003.

[6] Barham, P., et al. “Xen and the Art of Virtualization. .” *Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[7] Berger, Stefan, Scott McFaddin, Chandra Narayanaswami, and Mandayam Raghumath. “Web Services on Mobile Devices.” *Fifth IEEE Workshop on Mobile Computing Systems & Applications* (IBM T.J. Watson Research Center), 2003.

[8] Berman, F., G. Fox, and T. Hey. “Grid Computing: Making the Global Infrastructure a Reality. .” (Wiley & Sons) 2003.

[9] Bernstein, David, Erik Ludvigson, Krishna Sankar, Steve Diamond, and Monique Morrow. “Blueprint for the Intercloud: Protocols and Formats for Cloud Computing Interoperaibility.” *Fourth International Conference on Internet and Web Applications and Services*, 2009.

[10] Carroll, A., and G Heiser. “An Analysis of Power Consumption in a Smartphone.” *USENIX Annual Technical Conference*, 2010.

[11] Carter, Nicholas P. *Schaum's Outline of Computer Architecture.* 2002.

[12] Chen, Eric Y., and Mistukaka Itoh. “Virtual Smartphones over IP.” *NTT Information Sharing Platform Laboratories* (NTT Corporation), 2011.

[13] Chun, Byung-Gon, and Petros Maniatis. *Augmented Smartphone Applications Through Clone Cloud Execution*, 2009.

[14] Chun, Byung-Gon; Maniatis, Petro. “Dynamically Partitioning Applications between Weak Devices and Clouds.” *MCS*, June 2010.

[15] Chun, Byung-Gon; Maniatis, Petros. “Augmented Smartphone Applications Through Clone Cloud Execution.” *HotOS*, 2009.

[16] Clark, Henry. “DAWGS: A Distributed Compute Server Utilizing Idle Workstations.” *Journal of Parallel and Distributed Computing*, 1992.

[17] Dou, Adam, Dimitrios Gunopulos, Vana Kalogeraki, Taneli Mielikainen, and Ville Tuulos. “Misco: A MapReduce Framework for Mobile Systems.” *3rd International Conference on Pervasive Technologies.* New York: ACM, 2010.

[18] Erdil, D.C., M. J Lewis, and N. Abu-Ghazaleh. “Adaptive Approach to Information Dissemination in Self-Organizing Grids.” *Proceedings of ICAS*, 2005.

[19] Fahad, Aijaz, Syed Zahid Ali, Mazzamil Aziz Chaudhary, and Bernhad Walke. “Enabling Resource-Oriented Mobile Web Server for Short-Lived Services.” *9th International Conference on Communications*, December 2009.

[20] Forestiero, Agostino, Carlo Mastroianni, Emilio Leonardi, and Michela Meo. “Self-Chord: A Bio-Inspired P2P Framework for Self-Organizing Distributed Systems.” *Transactions on Networking* (ACM), October 2010.

[21] Foster, Ian. “Service Oriented Science.” (Math & Computer Science Division, University of Chicago) May 2005.

[22] *Google App Engine.* 2011. http://code.google.com/appengine/ (accessed 2011 йил 19-April).

[23] Gray, J., P. Helland, P. E. O'Neill, and D. Shasha. “The stages of replication and a solution.” *SIGMOD International conference on Management of Data.* ACM, 1996.

[24] *Hadoop website.* http://hadoop.apache.org/core/ (accessed 2011 йил 1-August).

[25] Hagman, Robert. “Process Server: Sharing Processing Power in a Workstation Environment.” *Conference on Distributed Computing Systems*, 1986.

[26] Hassan, Mahbub, Weiliang Zhao, and Jian Yang. “Provisioning Web Services From Resource Constrained Mobile Devices.” *3rd International Conference on Cloud Computing* (Department of Computing, Macquerie University), 2010.

[27] Hendricks, E. C., and T. C. Hartmann. “Evolution of a virtual machine subsystem.” *IBM Systems Journal* 18, no. 1 (1979).

[28] Hennessey, John. “The Future of Systems Research.” (IEEE) August 1999.

[29] IETF. *RFC for Unique Local IPv6 Unicast Addresses.* http://tools.ietf.org/html/rfc4193.

[30] Kambhatla, Srikanth, and Jonathan Walpole. “Recovery with limited replay: Fault tolerant processes in Linda.” (IEEE) 1990.

[31] Kennington, Jeff, Eli Olinick, and Dinesh Rajan. *Wireless Network Design: Optimization Models & Solution Procedures.* California: Springer, 2011.

[32] Kistler, J. J., and M. Satyanarayanan. “Disconnected operation in the Coda file system.” *Transactions on Computer Systems.* 1992.

[33] Kistler, J., and M. Satyanarayanan. “Disconnected Operation in the Coda File System.” *Transactions on Computer Systems* (ACM), 1992.

[34] Klein, Andreas, Christian Mannweiler, Joerg Schneider, and Hans Schotten. “Access Schemes for Mobile Cloud Computing.” *11th International Conference on Mobile Data Management.* IEEE, 2010.

[35] Kreger, H. *Web Services Conceptual Architecture.* 2001 йил May. http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf (accessed 2011 йил 11-March).

[36] Kung, H. T., et al. “Network-based Mutlicomputers: An Emerging Parallel Architecture.” *Supercomputing*, 1991.

[37] Li, Fulu, David P. Reed, and Andrew Lippman. “Collaborative Storage with Mobile Devices in Wireless Networks for P2P Media Sharing.” (MIT) 2008.

[38] Li, Kai, Richard Lipton, Richard DeWitt, and Jeffrey Naughton. “SHRIMP: Scalable High Performance Really Inexpensive Multicomputer Project.” *ARPA High Performance Computer Software*, September 1993.

[39] Liang, Hongbin, Dijiang Huang, Lin Cai, Xuemin Shen, and Daiyuan Peng. “Resource Allocation for Security Services in Mobile Cloud Computing.” *Workshop on M2MCN.* IEEE, 2011.

[40] Marinelli, Eugene E. “Hyrax: Cloud Computing on Mobile Devices using MapReduce.” *Carnegie Mellon University*, September 2009.

[41] Maymounkov, Petar, and David Mazieres. “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.” *New York University*, 2002.

[42] McQueen, Darren. “The Momentum Behind LTE Adoption.” *IEEE Communications Magazine* (IEEE), February 2009.

[43] Mummert, L.B., M.R. Ebling, and M. Satyanarayanan. “Exploiting Weak Connectivity for Mobile File Access.” *15th Symposium on Operating Systems Principles*, 1995.

[44] Murphy, Amy L., Gian Pietro Picco, and Gruia Catalin Roman. “Lime: A Middleware for Physical and Logical Mobility.” (IEEE) 2001.

[45] *Nimbus Home Page.* http://www.nimbusproject.org (accessed 2011 йил 22-April).

[46] NIST. 2011. http://csrc.nist.gov/groups/sns/cloud-computing (accessed 2011 йил 17-April).

[47] Nurmi, Daniel, et al. “The Eucalyptus Open-Source Cloud-computing System.” *Proceedings of Cloud Computing and Its Applications*, October 2008.

[48] *Open Nebula Home Page.* http://www.opennebula.org (accessed 2011 йил 24-April).

[49] Parkhill, Douglas. *The Challenge of the Computer Utility.* Addison-Wesley, 1966.

[50] Patterson, David A., and David R. Ditzel. “The Case for Reduced Instruction Set Computing.” *Computer Architecture News* (ACM) 8, no. 6 (October 1980).

[51] Peterson, K., M. Spreitzer, D. Terry, M. Theimer, and A. Demers. “Flexible update propagation for weakly consistent replication.” *Symposium on Operating System Principles.* ACM, 1997.

[52] Peterson, K., M.J. Spreitzer, M. M. Theimer, and A. J. Demers. “Flexible Update Propagation for Weekly Consistent Replication.” *16th Symposium on Operating Systems Principles* (ACM), 1997.

[53] *RFC 791.* 1981. www.ietf.org.

[54] Riva, Oriana, and Kanga Kangasharju. “Challenges and Lessons in Developing Middleware on Smart Phones.” *Computing Practices* (IEEE), October 2008.

[55] Rouston, Anthony, and Peter Druschel. “Pastry: Scalable, decentralized object location and routing for large scale peer-to-peer systems.” *18th Conference on Distributed Systems Platforms*, 2001.

[56] Sarathy, Vijay, Narayan Purnednu, and Rao Miffilineni. “Next Generation Cloud Computing Architectures.” *Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises*, 2010.

[57] Satyanarayanan, Mahedev, Paramir Bahl, Ramon Caceres, and Nigel Davies. “The Case for VM Based Cloudlets in Mobile Computing.” *Pervasive Computing* (IEEE), December 2009.

[58] Sempolinski, Peter, and Douglas Thain. “A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus.” *2nd IEEE International Conference on Cloud Computing Technology and Science*, 2010.

[59] Simoens, Pieter, Filip De Turck, Bart Dhoedt, and Demeester Piet. “Remote Display Solutions for Mobile Cloud Computing.” IEEE, 2011.

[60] Sobti, Sumeet, et al. “A Peer-to-Peer Mobile Storage System.” *Data Management & Storage Technology*, 2002.

[61] Stoica, I., R. Morris, D. Karger, M. F. Kaashock, and H. Balakrishnan. “Chord: A Scalable Peer-to-Peer Looking Service for Internet Applications.” *SIGCOMM* (ACM), 2001.

[62] Tanenbaum, Andrew S., and Martin Van Steen. *Distributed Systems, Second Edition.* (Prentice Hall), 2006.

[63] Tian, Ye, Kai Xu, and Nirwan Ansari. “TCP in Wireless Environments. Problems and Solutions.” *Letters, volume 9, no. 1*, 2005.

[64] Warner, Steven A., and Alexander F. Karman. “Defining the Mobile Cloud.” 2010 йил 16-August. www.nasa.gov/ppt/482352main_2010_Monday_1_Warner.Steven_r5.ppt (accessed 2011 йил 12-July).

[65] Wikman, Johan, and Forenc Dosa. “Providing HTTP Access to Web Servers Running on Mobile Phones.” *Nokia Research Centre*, May 2006.

[66] Xu, Andrew, and Barbara Liskov. “A Design for a Fault Tolerant, Distributed Implementation of Linda.” (IEEE) 1989.

7.2. Appendix 1: Future extensions to Icarus

The design of Icarus raises many challenges and also affords many opportunities to extend and enhance the system. To promote the ability to design a core prototype of Icarus within a short period of time it has not been possible to incorporate some of these enchancements. As a result the following list presents opportunities for extending Icarus in the future.

1. Node deployment: There are three main components to Icarus; client, directory and storage node functionality. For deployment Icarus would be designed such that each node is generic and any combination of the above functionality can be activated or switched off. The key deliverable of this design choice is that nodes can reside on a mobile phone or on a commodity server. A node hosted on a commodity server may only provide directory functionality for example.

2. Operating system heterogeneity. Given the heterogeneity of mobile operating systems, significant time will not be spent on deploying the design to different platforms. The use of Web services at the storage nodes would however provide greater interoperability. Web services have not been used for the prototype as it made system evaluation more complex.

3. Latency: The distributed mobile storage needs to be location transparent and location independent. However ideally it would minimize the latency associated with file retrieval. Two options are envisaged:

    a. The directory decides which copy of a file to recommend to the client based on the estimated latency in transmitting the file.

    b. The client is provided with a list of nodes that have a copy of the file and the associated round trip time of the nodes relative to a parent directory. A user of the system can then choose which copy to retrieve and has subsequent options should a node fail during transmission.

4. System state health checks: Ideally the system would utilize an efficient manner for directory nodes to determine the availability of storage nodes prior to advising a client of the location of nodes with the requested files. This could be implemented using UDP “check-in” packets. Two options exist:

    a. UDP packets could be sent periodically from the storage node to the directory node to acknowledge that the node is available.

    b. The directory node could send a UDP packet to the storage node at the time a client requests access to a file. This would reduce the volume of messages overall so that messages are sent only to nodes that may be currently required. This option would require these specific storage nodes to respond to acknowledge availability. A lack of response would indicate that the node was offline.

5. Proximity: Ability to use UDP check-in packets or GPS coordinates to determine latency in communicating with specific nodes. For example with GPS coordinates you could ensure that files are replicated across different geographic locations to reduce the probability of failure due to a significant telecommunications fault such as a fiber cable cut impacting a region of wireless base stations.

6. Self-healing: A process to ensure that if only one copy of a file exists given node churn, that the Icarus directory would seek to make further replicates of the file on other nodes. This requires a dynamic replication algorithm that will sense the state of the network and reconfigure the location of replicate files accordingly. In the event of partial network failure, directory nodes should identify files that require protection and re-replicate as necessary to surviving nodes. In essence the network should seek to ensure its files survive.

7. Replication optimization. A suitable algorithm that optimizes the number of replicate files that exist based on the size of the network.

8. Consistency: Icarus could be extended to allow for read and write access to information in the future. This would require appropriate locking of the files to ensure serialized access for consistency. A replication management system similar to Gossip could be potentially used to add such functionality. This would go beyond an

extension of Icarus as a mobile distributed storage system to a mobile distributed file system. Nonetheless it would require significant work in optimizing the locking and synchronization in order to manage consistency.

9. Access control: Implement the ability for mobile participants to require access control at Icarus node or content level. In both cases the requesting client may have to authenticate and provide a password to access the node or specific content on a node.

10. Anonymity: The contextual file name could be decoupled from the actual file. This would allow for files to be stored on nodes with a greater degree of anonymity. The directory nodes would be responsible for mapping the actual file names requested to a hash of the file value. This enhancement if implemented securely could provide a secure and highly distributed mobile infrastructure to support file distribution such as Wikileaks.

11. File management: Improvements to allow for full read / write access and collaboration using Icarus and to incorporate quora based voting to allow for an accurate response where some replicas are inconsistent.

12. Network load efficiency: Develop system so that it can identify when multiple users upload the same file, through the use of the file content hash and optimise the volume of replicas rather than allow users to flood the network with various copies of the same file. This could be particularly important for sharing popular file information.

7.3. Appendix 2: UML diagrams of Icarus

Diagram 19: UML diagram of simulation package

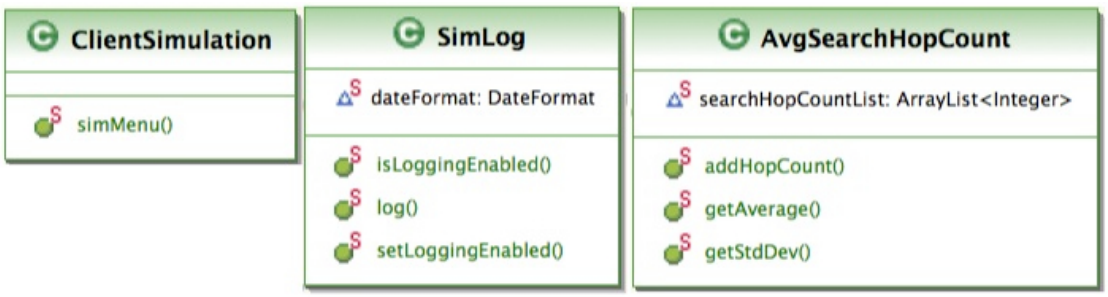


Diagram 20: UML class diagram - client package

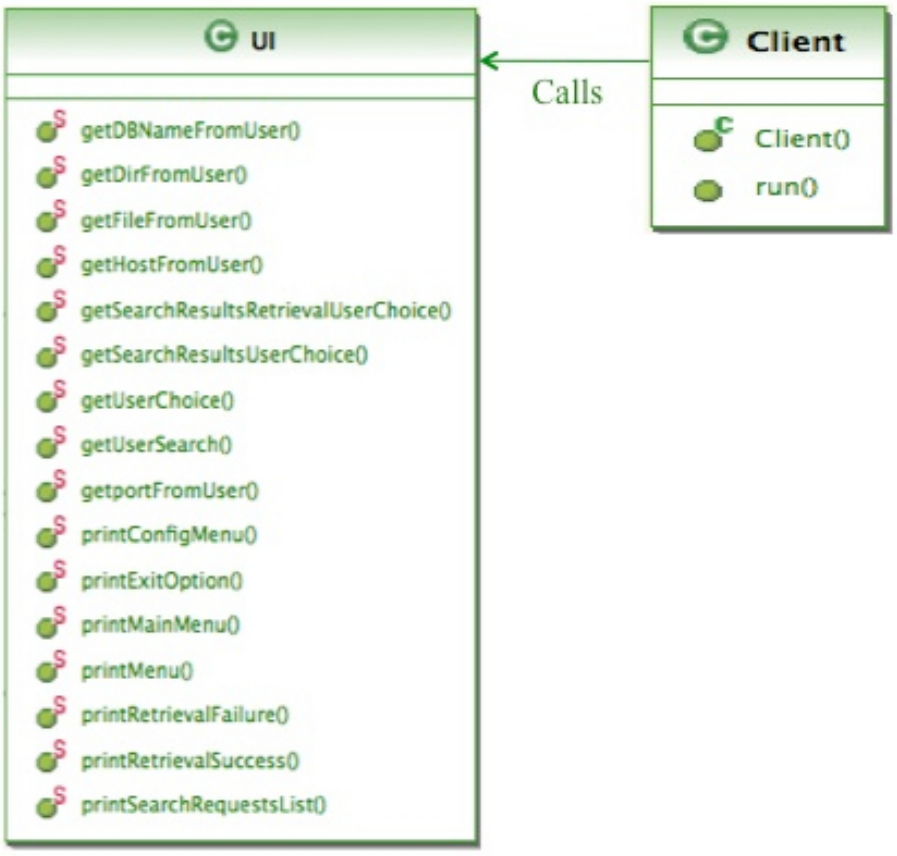


Diagram 21: UML class diagram - directory package

Diagram 22: UML class diagram - storage package

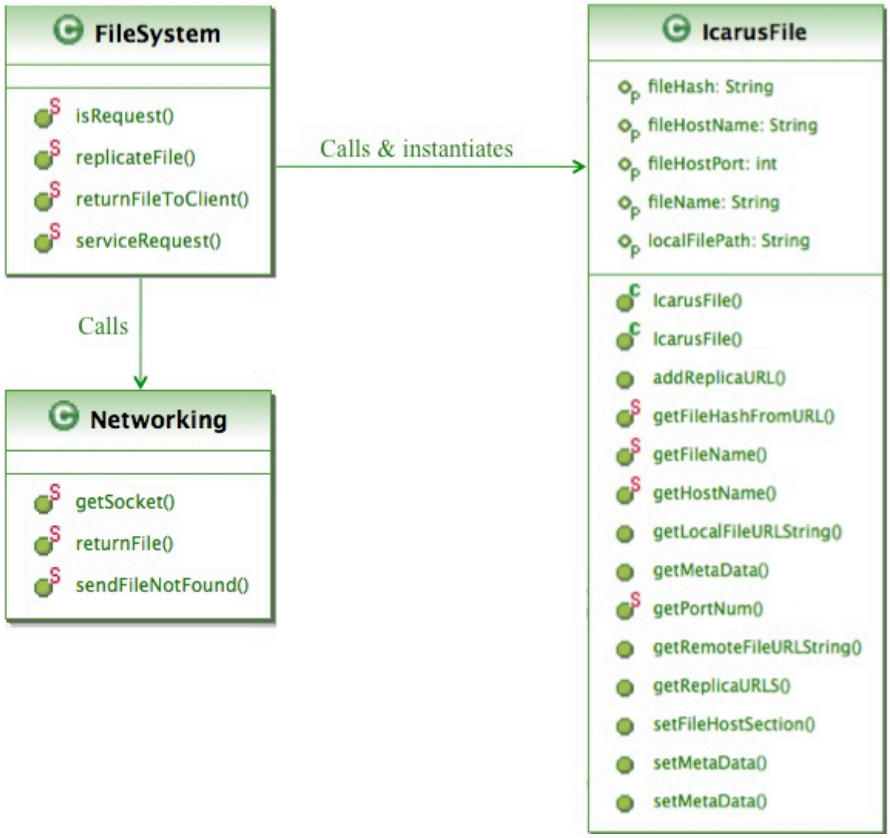
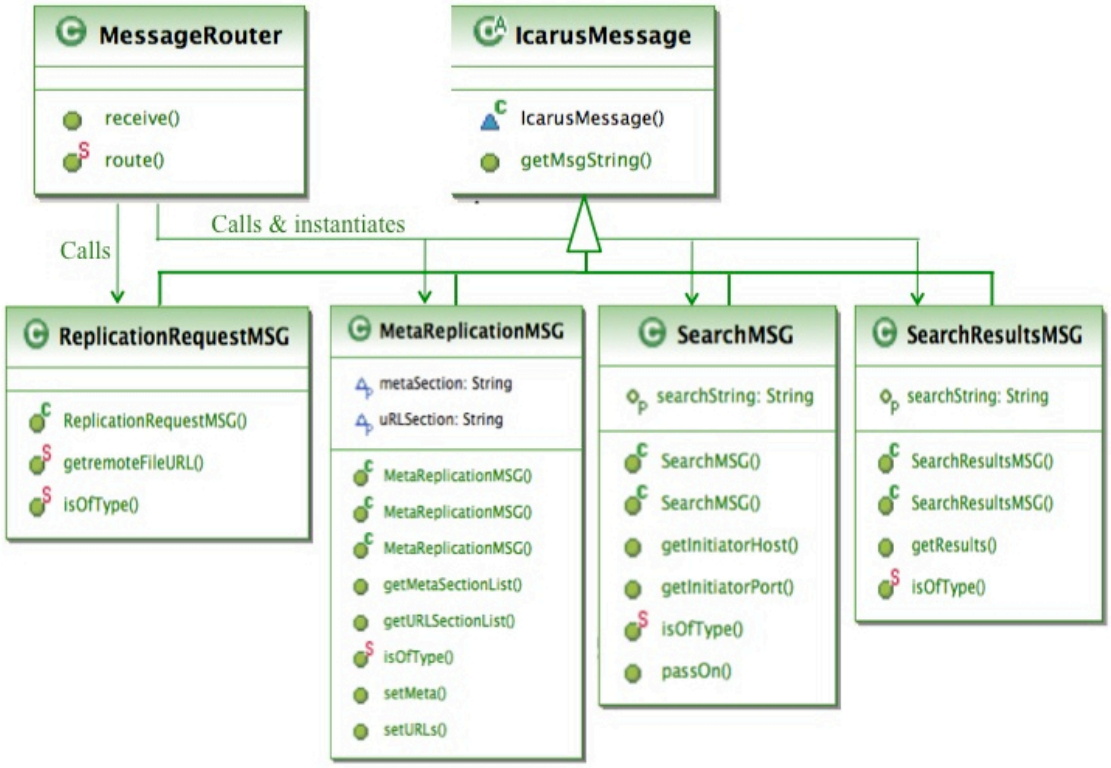


Diagram 23: UML class diagram - messaging package



7.4. Appendix 2: Icarus simulation – sample log output

2011/08/24 17:55:18 Icarus: New Simulation started ---------- simstart
2011/08/24 17:55:18 Network nodes: 10
2011/08/24 17:55:18 Iterations attempted: 10
2011/08/24 17:55:18 Nodes churning: 0
2011/08/24 17:55:18 Peers each node is aware of: 10
2011/08/24 17:55:18 Peer distribution: 1 (0 = ring, 1 = random)
2011/08/24 17:55:18 Search message hop count: 50
2011/08/24 17:55:18 Replication hop count: 3
2011/08/24 17:55:18 Meta replication count: 10
2011/08/24 17:55:18 Replication request hop count: 10
2011/08/24 17:55:18 Creating threads.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8081.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8082.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8083.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8084.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8085.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8086.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8087.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8088.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8089.
2011/08/24 17:55:19 Adding 10 random peer nodes for Icarus node 8090.
2011/08/24 17:55:19 Iteration count 0
2011/08/24 17:55:19 Sending file from 8085
2011/08/24 17:55:20 Trying to send message to 172.16.168.1:8086
2011/08/24 17:55:20 After message sent
2011/08/24 17:55:20 File replication to 8086
2011/08/24 17:55:20 Decrementing replication request, Passing on ReplicationRequest (count 10)
2011/08/24 17:55:20 Msg Router Passing on replication request
2011/08/24 17:55:20 Trying to send message to 172.16.168.1:8088
2011/08/24 17:55:20 After message sent
2011/08/24 17:55:20 Decrementing replication request, Passing on ReplicationRequest (count 9)
2011/08/24 17:55:20 Msg Router Passing on replication request
2011/08/24 17:55:20 Decrementing meta replication hopcount, Passing on MetaReplication (count 10)
2011/08/24 17:55:20 Msg Router Passing on meta replication message
2011/08/24 17:55:20 Trying to send message to 172.16.168.1:8088
2011/08/24 17:55:20 After message sent
2011/08/24 17:55:20 Trying to send message to 172.16.168.1:8087

…..

2011/08/24 17:56:14 File retrieval succeeded
2011/08/24 17:56:14 Success/Fail 10/0
2011/08/24 17:56:14 Nodes Iter Churn NodeAware NADist SearchHopcount Repcount Success Fail MetaReplications RepReqHops MetaReplicationHops SearchHopsTillResultAvg SearchHopsTillResultAvgStd
2011/08/24 17:56:14 10 10 0 10 1 50 3 10 0 10 10 10 2.1 1.0440306508910595

2011/08/24 17:56:14 Icarus: Simulation ended.