Amorphous OS

a new paradigm for a computing environment OS

"Water is insubstantial. By this I mean you can not grasp
hold of it. You can not punch it and hurt it. Be formless, shapeless, like water. Now you put water into a cup, it becomes the cup, you pour water into a bottle, it becomes the bottle, you put water into a teapot, it becomes the teapot. Now water can flow or it can crash...be water my friend." --Bruce Lee

A colony of cells cooperates to form a multicellular organism under the direction of a genetic program shared by the members of the colony. A swarm of bees cooperates to construct a hive. Humans group together to build towns, cities, and nations. These examples raise fundamental questions for the organization of computing systems:

How do we obtain coherent behavior from the cooperation of large numbers of unreliable parts that are interconnected in unknown, irregular, and time-varying ways?

What are the methods for instructing myriads of programmable entities to cooperate to achieve particular goals?

These questions have been recognized as fundamental for generations. Now is an opportune time to tackle the engineering of emergent order: to identify the engineering principles and languages that can be used to observe, control, organize, and exploit the behavior of programmable multitudes.

We call this effort the study of amorphous computing.

USERS Perspective

Why does the user need to see the local hardware?

With modern networking we can eliminate the users awareness of disk space, memory size, cpu power.

We can even go far enough to eliminate file names, paths and directories, and even network topologies.

These concepts are as irrelevant to reading E-mail or browsing the web as head cylinder and sectors on the hard drive are.

Developers Perspective

You should be able to write code in what ever is most familiar, comfortable and gets the job done.

Dissimilar languages should have a common method for communication.

Interaction between languages, CPU's, OS's, systems, and remote locations can be

Direction of computing.

Most aspects of computing are improving exponentially.

(CPU, RAM, DRIVES, NETWORKS, WIRELESS)

For the past 20 years, a system at that time has been able to emulate a system 5 years older. Moores Law stated 2x in 18 Months, this is 10x in 5 years. and 100x in 10 Years.

Soon Any wrist watch can run anything we have today faster then we can now!.

ASP and Network Software Distribution

The Current architectures make this very impractical.

Installing massive bloat ware packages are time consuming, risky of crashes, often require reboots and is difficult to copy protect.

The ASP's are limited to software installs or the web interface.

A new model for incremental payments and just in

Castles in the Sky

This all sound great but how can this be done?

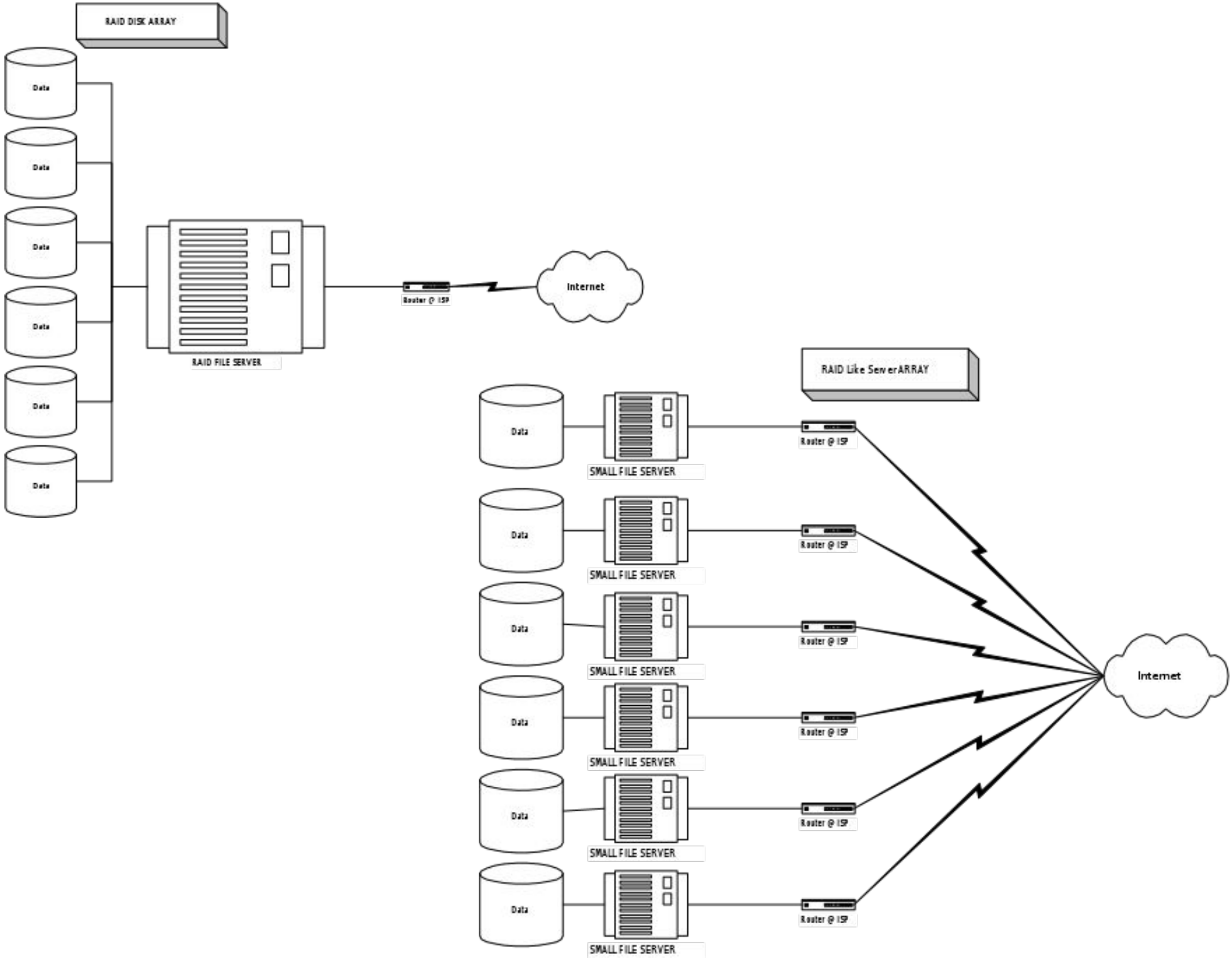
Amorphous file System

A Decentralized file system

- Extend one continuous virtual file system across multiple networked computers and operating systems, even the Internet.

- Eliminate the concept of directories. As they are currently know.

- Any file may be anywhere in physical space on the computers. Each computer locates and re-locates files and parts of files transparently on the fly, this is hidden by an abstraction layer.


RAID DISK ARRAY
Data
Data
Data
Data
Data
Data
RAID FILE SERVER
Router @ ISP
Internet
RAID Like Server ARRAY
Data
SMALL FILE SERVER
Router @ ISP
Internet

- Eliminate all flat plain old files. All files are now Objects. Meaning all have headers, contain a preamble with the files history, revision history, creation history including which computer and user worked on it. Copy History and tracking! Links to the original and an option to delete the data for the copy and just keep the links. An encrypted security access control section, usage history. And a package history. Files are categorized as a user created, or part of an application, or part of the Operating system.

- There is only one amorphous file system, but within it multiple virtual file systems are created and abstracted. Each program is confined to it's own individual file system and can not access anything external to it. Each user's data resides in it own file system. A user can not see kernel or program file systems just his own. Each user's session only mounts the file systems needed.

- Each file system is an object.

- The amorphous file system can be just an abstraction on top of UFS or MSDOS_FS or actually run native of the drive.

- All programs are a collection of objects and classes. Each UN-file now may contain several objects internal to it, images programs data, etc. even several program

Software Architecture

Replace legal “copyright” with a profit or “compensation right”. Meaning anyone can copy and transport and use the software but the author reserves the right to profit from his work. This ties into an incremental, fractional micropayment system for one of several profit models.

Micropayments for:

- Code or Data usage
- server CPU usage
- drive space usage

All data and code are now objects.

An object may contain both code and data.

An object as all the knowledge it needs.

An object is only cached locally unless it's been created locally.

Objects reside at a real network address similar to a URL to a java class.

Objects communicate via a Message passing interface and could go through an object request broker. Object may run remotely (RPC) or be transported locally to run like Java classes. This should be completely transparent.

What do I mean by Object?

In Unix script are like objects.

#!/bin/sh

code ...

data ...

The /bin/sh tells the shell(os) what program is to be used to process the rest of the file.

Unix doesn't care what's in the file. Perl, TCL, csh, bash, or even uuencoded data, or a jpeg image even.

Format of Amorphous objects

Headers

- author info
- authority
- source url, cache url's
- change info

code or data

(encapsulated in mime type headers)

- content type: perl/script
- content length: 4291 bytes

or

- content type: 586/binary

or

Code to deal with GUI and calling functions.

Addbutton(loc/size, icon, object, parameters to pass)
This is an alternative to event driven.

CallObject(object, send variables, return variables)
This may be blocking or non-blocking(a thread)

Where object can contains a URL like path to the objects original source location. This can be cached at several posible locations and/or executed on any number or locations.

This is more like html and java in some ways

Now an HTML page could launch objects!

see manual

or

see bla bla spread sheet

or from basic, C, C++, Java, Perl or any other language.

Addbutton(loc,"http://www.livecam.com");

Another advantage

How to allow GUI based code to take advantage of parallel processing

How to simplify object oriented to run on parallel processing clusters.

SMP will soon be integrated into a multiple CPU in one chip package, Applications need to handle this gracefully.

The Next level

Whole desktop with applications available on a subscription basis from a yahoo or google.

Software, Books, music, video, photos, etc.

Backup services, remote repair and upgrades and virus's will be a thing of the past.

Your desktop work environment will be more like a home page. It's not tied to a computer, since the hardware is disposable. It's

tied to your portal and user account