Cloud Operating System Using Node.js

Mr. Praful Surve[1]Mr. Vatsal Shah[2] Miss. Surbhi Shah[3]Mr. S.P. Khachane [4]

[1,2,3]Student [4]Professor

[1,2,3,4]Rajiv Gandhi Institute of Technology, Mumbai

Abstract—the idea behind Cloud Operating System is that the whole system lives in the web browser. The client must have only a web browser to work with Cloud Operating System and all its applications. This applies to for both modern and obsolete PC's An Open Source Platform designed to hold a wide variety of Web Applications. Cloud Operating System would be a new definition of an Operating System, where everything inside it can be accessed from everywhere inside a Network. All you need to do is login into your Cloud Operating System server with a normal Internet Browser, and you have access to your personal desktop, with your applications, documents, music, movies... just like you left it. Cloud Operating System would let you to upload your files and work with them no matter where you are.

Keywords:- Cloud operating system, cloud process, node.js

I. INTRODUCTION

In recent years a new technological paradigm has emerged in the computer science research community. This paradigm is referred to as Cloud Computing, a new computing paradigm whose aims are to allow users to utilize computing infrastructure over the network, supplied as an on-demand service. There are already many commercial solutions based on this approach, as well as many academic research projects that are especially focused on resource management of an infrastructure that comprises computational power, storage space and communication networks.[1][2]

In particular, the demand of storage capacity is increasing, determined by, for example, scientific (e.g. physics experiments) and commercial (e.g. e-commerce sites, search engines) applications, that produce huge quantity of data. Building a data storage service that is pervasive, available, and scalable and that can handle massive quantities of data has always been a priority for every distributed system paradigm and infrastructure. Cloud Computing has different goals and characteristics and thus poses new challenges in this area of research.

Cloud Computing applications, especially data intensive ones, typically need storage services that provide large amount of storage capacity and the ability of retrieving stored data at any time and in any condition (e.g., infrastructure component failures). Moreover, services are hosted and data are located on third-party resources, a condition that poses a threat to data confidentiality. In many cases, applications or system-specific functions require the availability of raw block devices, for instance when a specific file system is necessary, or when the application needs to directly access physical storage (e.g., in the case of a DBMS).[3]

Like a Server Operating System (OS), a Cloud OS is responsible for managing resources. In a server, the OS is responsible for managing the various hardware resources, everything inside a server's chassis. It hides the hardware operation details and allows these scarce resources to be efficiently shared. A cloud OS serves the same purpose. Instead of managing a single machine's resources, a cloud OS is responsible for managing the cloud infrastructure, hiding the cloud infrastructure details from the application programmers and coordinating the sharing of the limited resources. But unlike a traditional OS, a cloud OS has to do everything at scale. IBM CEO Thomas J. Watson is well known for his 1943 statement, "I think there is a world market for maybe five computers." Although it is often laughed at since the advent of Personal Computers, it is becoming a reality again. The only difference is that we refer to these computers as clouds. Today, only a handful of companies, such as Google, Microsoft, Amazon and Yahoo, need and are capable of building a cloud–a large server farm with hundreds of thousands of servers. For example, it is reported that Google has well over 1 million servers. Managing such big infrastructure requires the OS to be extremely scalable.

Cloud Operating System is an open source web desktop following the cloud computing concept. It is mainly written in PHP, XML, and JavaScript. It acts as a platform for web applications written using the Cloud Computing concepts. It includes a Desktop environment with number of applications and system utilities. It is accessible by portable devices via its mobile front end. Every Cloud Operating System lets you upload your files and work with them no matter where you are. It contains applications like Word Processor, Address Book, PDF reader, and many more developed by the Cloud vendor.

Cloud is a simplified Operating System that runs just on a Web browser, providing access to a variety of web-based applications that allow the user to perform many simple tasks without booting a full-scale Operating System. Because of its simplicity, Cloud can boot in just a few seconds. The Operating System is designed for Net books, Mobile Internet Devices, and PCs that are mainly used to browse the Internet. [2]

II. TECHNICAL VIEW

A. *Objectives*

1) Being able to work from everywhere, regardless of whether or not you are using a full-featured, modern computer, a mobile gadget, or a completely obsolete PC.
2) Sharing resources easily between different work centers at company, or working from different places and countries on the same projects.
3) Being able to continue working if you have to leave your local computer or if it just crashes, without losing

data or time: Just log in to your Cloud Operating System from another place and continue working.
4) Worldwide availability of Cloud Operating Systems: As it's available through Internet
5) Requires only Web Browser for accessing Cloud Operating System.
6) Browser and Platform independent.

B. *System structure*

The Existing Cloud OS system available connects the client to the server, and the server treats each request as an independent thread. The Server then performs thread scheduling for all the threads that are requested by every client on the network. Since Cloud Systems are widely used now-a-days, there is lot of load on server residing on the cloud. Since every client request is treated as independent thread, each thread is rescheduled as per the priority, using Apache Web Server or Fast CGI technology. What if large number of client connects to the server, in our case large number of users hitting request for Cloud OS. This would definitely overload the server and there might be possibility of crashing the server. [2]

In our system, we replace the Multi-threaded Web Server with Single-threaded system with the help of Node.js. This in turn reduces the load from server and ensures high scalability on Server.

Node.js is a server-side software system designed for writing scalable Internet applications, notably web servers. Programs are written on the server side in JavaScript, using event-driven, asynchronous I/O to minimize overhead and maximize scalability.

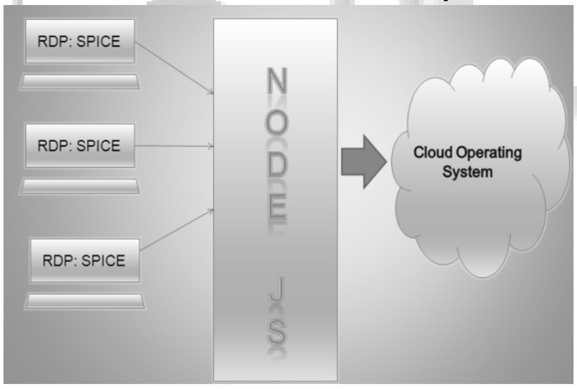

Fig. 1:System structure of Cloud Operating System

Node.js is a server-side software system designed for writing scalable Internet applications. Programs are written on the server side in JavaScript, using event-driven, asynchronous I/O to minimize overhead and maximize scalability.

Node.js creates a web server by itself, making it unnecessary to use web server software such as Apache or Lighted and allowing full control of how the web server actually works. Node.js enables web developers to create an entire web application on both server-side and client-side in one language (JavaScript).

Node.js is a packaged compilation of Google's V8 JavaScript engine, the libUV platform abstraction layer, and a core library, which is itself primarily written in JavaScript.

C. *Using Node.js*

Node.js is a server-side software system designed for writing scalable Internet applications, notably web servers. Programs are written on the server side in JavaScript, using event-driven, asynchronous I/O to minimize overhead and maximize scalability.

Node.js creates a web server by itself, making it unnecessary to use web server software such as Apache or Lighttpd and allowing full control of how the web server actually works. Node.js enables web developers to create an entire web application on both server-side and client-side in one language (JavaScript). Node.js is a packaged compilation of Google's V8 JavaScript engine, the libUV platform abstraction layer, and a core library, which is itself primarily written in JavaScript. [6]

Node.js was created by Ryan Dahl starting in 2009 and its development and maintenance is sponsored by Joyent, his former employer Dahl's original goal was to create web sites with push capabilities as seen in web applications like Gmail. After trying solutions in several other programming languages he chose JavaScript because of the lack of an existing I/O API. This allowed him to define a convention of non-blocking event-driven I/O.

There are four building blocks that constitute Node. First, Node encapsulates libuv to handle asynchronous events and Google's V8 to provide a run-time for JavaScript. Libuv is what abstracts away all of the underlying network and file system functionality on both Windows and POSIX-based systems like Linux, Mac OS X and UNIX. The core functionality of Node, modules like Assert, HTTP, Crypto etc, reside in a core library written in JavaScript. The Node bindings, written in C++, provide the glue connecting these technologies to each other and to the operating system. [6]

We can use Node.js for creating Cloud Web Server, where we can implement Socket Programming. In our system, client request for the TTY command to execute, the request is initiated as Socket Programming event, the event is received by Node.js Web Server. At Node Server, the event are scheduled as per the Node.js Scheduler within a single thread i.e. node process. The Server OS receives the event as the TTY command which is executed on server space. After the complete execution of the command by TTY process, the resultant output is hit back to Node.js Web server via sockets. The Web Server then forwards this result back to client as HTML data.

D. *Socket.IO*

Socket.IO is a JavaScript library for realtime web applications. It has two parts: a client-side library that runs in the browser, and a server-side library for node.js. Both components have a nearly identical API. Like node.js, it is event-driven.

Socket.IO primarily uses the Web Socket protocol, but if needed can fall back on multiple other methods, such as Adobe Flash sockets, JSON polling, and AJAX long polling, while providing the same interface. Although it can be used as simply a wrapper for Web Socket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.[6]

E. Read-Eval-Print-Loop (REPL)

A Read-Eval-Print-Loop (REPL) is available both as a standalone program and easily includable in other programs. The REPL provides a way to interactively run JavaScript and see the results. It can be used for debugging, testing, or just trying things out. By executing node without any arguments from the command-line you will be dropped into the REPL. It has simplistic emacs line-editing.

III. ARCHITECTURE

The Figure below explains the Logical Structure of Cloud Operating System. We use TTY process to execute on server by Socket Programming for client side request.

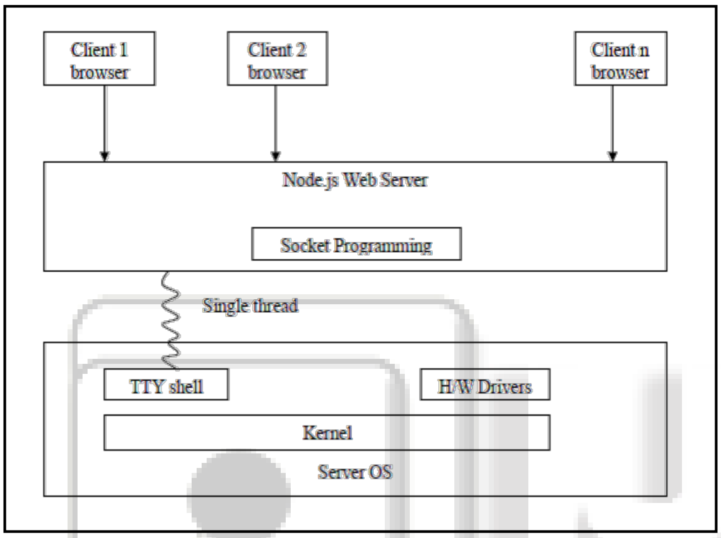

Fig. 2: Architecture of Cloud Operating System

In our system, client request for the TTY command to execute, the request is initiated as Socket Programming event, the event is received by Node.js Web Server. At Node Server, the event are scheduled as per the Node.js Scheduler within a single thread i.e. node process. The Server OS receives the event as the TTY command which is executed on server space. After the complete execution of the command by TTY process, the resultant output is hit back to Node.js Web server via sockets. The Web Server then forwards this result back to client as HTML data. [5]

We define the Cloud object as a set of local OS processes running on a single node, which are wrapped together and assigned locally a random identifier of suitable length to minimize the risk of system-wide ID collisions. A Cloud process (CP) is a collection of Cloud objects that implement the same (usually distributed) application.

IV. PERFORMANCE

Connecting to the company's apps from a private cloud sounds a very attractive option, but is it as smooth as a traditional client-server user experience?

First of all, most of the frontend heavy lifting takes place within the local browser: moving the mouse, minimizing or maximizing a window, navigating across the menus, typing, etc. are local tasks. The user will experience no latency in the most common and repetitive tasks of his or her day-to-day routine.

Secondly, Cloud Operating System architecture could clearly separates main tasks, with the possibility of dedicating separate servers if the load becomes intense.

Finally, Cloud Operating System kernel could be compiled in C++ for maximum speed. The result is outstanding: users won't notice that their Cloud Operating System desktop is actually on the company's private cloud and not on their local PC.

V. SECURITY

A. Philosophy

Cloud Operating System' philosophy is based on two concepts: defense in depth and separation of duties. Our philosophy is to reduce public services, provided it is configured with a good default security policy. In the event of doubt, several tools can be provided to the end user to decide the security level that they want to give their Cloud Operating System.

B. Management & Communication

The virtual appliance has host-based protection, with zero outside dependency. For that reason, the only channels for managing the solution are via SSH and the Admin Panel in Cloud Operating System. The Admin Panel can contain tools for diagnostic, activation of automatic updates, configuration of own SSL certificates, backup creation management, etc.

REFERENCES

[1] A. S. Tanenbaum and S. J. Mullender, An Overview of the Amoeba Distributed Operating System, Operating System Review.

[2] M. Ahamad, P. Dasgupta, and R. J. LeBlanc, Fault-Tolerant Atomic Computations in a n Object-based Distributed System, Distributed Computing Journal.

[3] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, Mach: A New Kernel Foundation for UNIX Development, Proceedings of the Summer USENIX Conference.

[4] M. Ahamad and N. E. Belkeir, Using Multicast Communication for Dynamic Load Balancing in Local Area Networks, ln 14th Annual Conf. on Local Computer networks.

[5] E. Allchin, Architecture for Reliable Decentralized Systems, Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology, 1983. (Available as Technical Report crr-rcs-83/23.)

[6] http://nodejs.org/api/os.html