World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:5, 2009

Toward Community-Based Personal Cloud Computing

Weichang Du

***Abstract*—**This paper proposes a new of cloud computing for individual computer users to share applications in distributed communities, called community-based personal cloud computing (CPCC). The paper also presents a prototype design and implementation of CPCC. The users of CPCC are able to share their computing applications with other users of the community. Any member of the community is able to execute remote applications shared by other members. The remote applications behave in the same way as their local counterparts, allowing the user to enter input, receive output as well as providing the access to the local data of the user. CPCC provides a peer-to-peer (P2P) environment where each peer provides applications which can be used by the other peers that are connected CPCC.

***Keywords*—**applications, cloud computing, services, software.

I. Introduction

THE users of distributed communities benefit from the ability to communicate and share information over the Internet [5]. An example of such a community would be a simple instant messaging network. Each user of the network has a group of friends whom they communicate with. Each user of an instant messaging network is a member of multiple communities. Friends of the user have their own independent contact lists, each forming a users' distributed community.

Existing applications for distributed communities revolve around communication and file-sharing. Networks such as ICQ and MSN Messenger are a popular way to build communities of users who wish to exchange conversations with one another. The file sharing networks such as eMule attract users who wish to share their files with others. In this paper, we propose a new application for distributed communities based on the cloud computing concept [1,2,9], which we call community-based personal cloud computing (CPCC). CPCC is a new model of cloud computing which provides an effective way for a distributed community to use shared their application resources. The users of CPCC would be able to share their personal applications with others. This is different from file-sharing since the users do not download applications from other users. The CPCC model simply allows users to execute shared personal applications remotely, unlike conventional cloud computing where remote applications run on application servers controlled by service providers.

Weichang Du is with Faculty of Computer Science, University of New Brunswick, Canada E3B 5A3. Phone: 506-458-7278; Fax: 506-453-3566; e-mail: wdu@ unb.ca).

There are two models available for building distributed communities. Each of these models provides a way to connect a group of computers with one another. Once connected the computers on the network are aware of one another and are able to share resources across the network. The distributed community is built on top of such a network. In a distributed community the users are concerned with communication between one another and do not need to be aware of the underlying workings of the network technology which they are using.

The first model is based on client-server architecture. In this architecture machines on the network play different roles. One machine is designated as a server and provides its resources to the others which are designated as clients. In a client-server model members of a community all connect to a central server which facilitates the communication between them. A web forum would be an example of a client-server based community [8]. The web server is responsible for providing all the services necessary for the community to function. The users of the community do not share their own computing resources with others. A distributed community can be built on top of a client-server model where users would be able to upload applications to a central server. Once the applications are installed on the server other users of the community can execute them remotely on the server. This is basically the same as the conventional cloud computing model.

The second model is based on a peer-to-peer (P2P) architecture. By contrast in the P2P model all peers are equal, cooperating with one another. Each machine on the P2P network provides a service to other peers on the network. This service can be used by any other machine on the network. An example of such a community is a P2P based file sharing network [6]. In this community each user serves files to other users. In a P2P based CPCC distributed community each user would designate which applications they wish to share with other users. Each peer would notify other peers of which applications it is sharing. Each user would then see the applications shared by the other users and be able to execute them. The applications execute remotely, on the peers where the applications are installed. In the P2P model the users build a community by sharing application resources of their individual computers. P2P architecture possesses several advantages over the client-server architecture for building personal cloud computing communities. This is because each user acts as a peer and is able to retain the applications which they wish to share on their local machine. In a client-server model the user would have to install the applications on a community server to make them accessible to other users. So

in a scenario where a user is a member of multiple communities he or she would be forced to provide a separate copy of the application for each of the community servers. By contrast in a P2P model the user would simply share his or her locally installed application with the communities which he or she is a part of.

Conventionally, when an application is installed on a computer, the application is only available to the users of that computer. The purpose of distributed communities is to allow users to communicate and share resources. In a distributed environment groups of users participate in increasingly complex interactions. Therefore it is desirable to be able to share complex resources easily. A common task in a community environment is for users to share their documents with one another [4]. When a user chooses to share a document with the community, others need a way to view this document. In order to be able to view documents created with a specific application everybody must have access to that application. This means that all the members of the community must have the same application available to them. When members of a community are not able to share applications, each user would have to have a copy of that application. That however is not always possible for a number of reasons. One reason may be that the users have different computing environments, and the applications may not be available for every system. Another problem could be that the users may not own the required applications. These factors hinder the ability of the members of a community to share documents.

Applications are not among those resources which can be shared in existing distributed communities [10]. Ideally the members of a community would have a common software desktop available to all the users, not matter what types of personal computers they use. This desktop would contain a collection of applications shared by the members of the community. Each member would have an option of making their local applications available to others. Once a user makes their applications available, other users of the community desktop would see those applications on their desktop and be able to use them.

Existing cloud computing technologies provide support for certain aspects of the concept of a community desktop, however these technologies are not designed to provide a community desktop. This is because these technologies have different goals or they are too low level. Technologies such as grid computing [9] and application, software, platform and infrastructure as services [3] have a different purpose from the CPCC model. These technologies are not concerned with building a community application sharing desktop. In Grid computing the concern is with distributing computation between multiple computers in order to speed up computation, while application services are concerned with remote application management. Technologies such as remote execution do not provide an end user solution. Instead they provide foundation for more sophisticated solutions. Remote execution technology, only allows remote execution of applications. This technology does not provide an application or user level solution for a distributed community desktop.

The objective of this paper is to investigate distributed personal application sharing for distributed communities. In essence personal application sharing is the ability for members of a community to make their applications usable by other members of their community. What this means is that if a user installs an application locally he or she can then make this application available for other users to use. The users executing remote applications do not need to install the shared applications locally on their machines. This is similar to users sharing files on the network. Shared personal applications should behave in the same manner; when a user shares an applications other users would see it on their desktops and be able to execute it as if it was a local application.

The rest of the paper is structured as follows. Section 2 describes the functional requirements of CPCC. Section 3 describes the architecture of CPCC and the various components of the system. The implementation of our prototype CPCC system is discussed in Section 3. Section 4 evaluates the prototype implementation and discusses the testing results. Section 5 gives concluding remarks.

II. Community-based personal Cloud computing model

In CPCC, the P2P component is to provide the functionality necessary to maintain a P2P network. The peers must be able to register and resign with a CPCC community. The registered peers must be able to sign on and off from the community. This functionality is provided by the CPCC network. The CPCC community is built on top of the CPCC network. The CPCC network consists of a group of registered peers. The member of each peer in the CPCC community provides applications for the other peers.

To join the CPCC community, the administrator of a peer must register it to become a member of the underlying CPCC network. Conversely the administrator can withdraw the peer membership from the CPCC network, hence the CPCC community. After a peer has become a registered member of a CPCC network, it will automatically join the network as an active member when it is online. When the peer goes offline it will automatically leave the network. The members of the CPCC network can be in two states, which are active and inactive. An active peer is a peer which is a member of the network which is online. An inactive member is one which is a member of the network but is not currently online. When a new peer joins the CPCC network or an existing member leaves the network, the rest of the peers must update the list of active members to reflect that change.

The members of the CPCC community have two roles. The first role is as an application provider that provides applications to be shared by the other members. The second role is as an application user. The application users make use of the applications provided by the other members.

An application provider must have control over how its applications are shared. The provider controls how many instances of a shared application can be running at one time.

The users of a CPCC community must be able to see shared applications provided by other members and request that the peers run shared applications.

In order to share its applications, the application provider must be able to advertise which applications it is sharing along with the usage policy on these applications. Once an application user requests an application the application provider must be able to execute the application on the provider's machine, send the application output to the user's machine and receive the user input and send it to the application. The application running on the provider's machine must also be able to access the files local to the user's machine during its execution.

When the application is running, the I/O must be setup between the application and the user, such that the user is able to receive the application output, and the application receives the user input. The application can be terminated in two ways. The standard way is for the application user to finish using the application and terminate it.

When the user terminates the application, the I/O channel must be closed and the application terminated by its provider. In the second scenario, the application provider chooses to terminate the application. In this case the application user must be informed that the application will become unavailable and given a chance to save their data before the application terminates. The number of running instances of the application is updated and peers are notified that the application becomes available for use by other members of the community.

When multiple copies of an application are shared by multiple members, the application user needs to be able to choose the appropriate peer to run the application. A load balancing mechanism is needed in order to choose on which peer the application should be run. This decision is affected by the latency of the connection of a peer and the system load of the peer. These factors must be taken into account when choosing a peer to run the application.

III. Community-based personal Cloud computing architecture

The CPCC architecture consists of three major subsystems. These sub-systems are shown in Figure 1. The DAS (Distributed Application Sharing) subsystem is responsible for handling the application sharing functionality. It handles the display and management of remote applications, provides local applications for remote peers, and manages the status of the shared applications. The P2P subsystem is used by the administrator to manage the community memberships. The Messaging subsystem is used for sending messages to other members of the community as well as forwarding the messages from other members to the appropriate components.

A. Messaging Subsystem

The messaging subsystem of the CPCC architecture is responsible for processing incoming and outgoing messages for each peer. The messages are used by the peers to communicate with other members of the community. The messaging subsystem consists of the Message Handler and the Peer Notifier components. The Message Handler is responsible for processing the incoming messages from the other peers on the network. These messages are forwarded to the other subsystems of the peer based on their types. The Peer Notifier is responsible for forwarding the outgoing messages, generated by the other subsystems of the peer, to the other peers on the network.

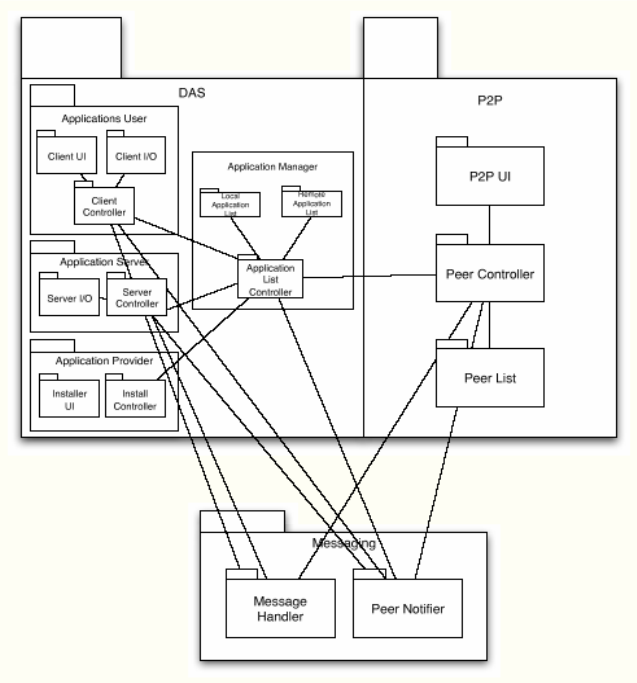

Fig. 1. CPCC architecture

B. P2P Subsystem

The P2P subsystem of the CPCC architecture is responsible for maintaining a consistent network of interconnected peers. This subsystem contains three components, the P2P UI, the Peer Controller and the Peer List.

The P2P UI allows the administrator of the peer to join and leave the DAS network. The administrator is also able to view the status of the network members using the P2P UI.

The Peer List keeps a list of the peers which are members of the DAS network. Each member can be either active or inactive. This is used to indicate if the peer is currently connected to the network.

The Peer Controller is the core of the P2P subsystem. It is responsible for processing the member registration and resignation, as well as monitoring the synchronization information. When individual peers become available or unavailable the Peer Controller must process the notifications sent by those peers and update their status in the Peer List accordingly. When this peer signs off the P2P Controller sends a notification to the rest of the peers. When a new member joins the network, the P2P Controller is responsible for synchronizing the Peer List. Finally the Peer Controller synchronizes with the Application List Controller in the DAS subsystem to notify it which peers are active, or inactive.

C. DAS Subsystem

The DAS subsystem of the CPCC architecture controls and manages the CPCC related activities. These activities include

installing, running, listing and sharing the applications on the CPCC network. The DAS subsystem consists of the Application User, Application Server, Application Provider and the Application Manager components.

The Application User component provides the interface as well as the functionality necessary for accessing remote applications shared in the CPCC community. The Application User consists of the Client UI, the Client Controller and the Client IO subcomponents.

The Client UI provides the user interface for the application user of a peer. It lists the shared applications currently available in the CPCC community and allows the user to execute those applications.

During the execution of an application the Client IO links up with the Server IO of the peer providing the application. It is used to provide a method for the remote applications to display the output and receive the user input and data.

The Client Controller is responsible for processing the user requests forwarded by the Client UI. It also establishes and manages the client side of the remote application execution.

The Client Controller retrieves the application information from the Application List Controller and sets up the Client IO for use by the remote applications. During the execution of the application the Client Controller monitors the execution and termination of the application by either the local user or the remote peer. When the user or the remote peer closes a remote application the Client Controller destroys the Client IO channel for that application.

The Application Server component provides the applications for the other members of the community. The Application Server consists of the Server IO and the Server Controller sub-components.

The Server IO provides the IO channel which is used by the application to send the data to the client. This channel links up with the client IO channel of the client peer.

The Server Controller processes the requests from the remote client peers to execute the local applications. The Server Controller creates the server IO and initiates the control and monitoring of the execution and termination of the local application. It processes the client termination requests and synchronizes the status of the application with the Application List Controller component.

The Application Provider component provides for the administrator to install and uninstall shared applications. The Application Provider consists of the Installer UI and the Install Controller sub-components.

The Install UI provides the interface necessary for the administrator to install and manage the applications provided by the peer.

The Install Controller controls the installation and un-installation of applications on the local peer which are to be shared with other peers. It notifies the Application List Controller that a new application has become available when the administrator installs an application, and that an application is no longer available when an application is uninstalled.

The Application Manager component is responsible for storing and managing the list of applications which are available on the DAS network as well as managing the list of locally shared applications. This component consists of the Application List and the Application List Controller sub-components.

The Application List is a container which keeps a list of available applications as well as their status. It also stores the list of locally shared applications.

The Application List Controller is responsible for ensuring that the Application List is up to date when applications become available and unavailable as well as when applications are installed or uninstalled.

IV. Prototype Implementation of Community-based Personal Cloud Computing Architecture

A prototype of the CPCC architecture has been implemented. The CPCC prototype is implemented in the Linux environment. Linux was chosen due to its network oriented nature. Linux is designed to be a multi-user system, meaning that multiple users are able to connect to and run applications on a Linux workstation either locally or remotely. The X Window system is designed to be used in a network centric environment. An X application can be executed remotely or locally. The network file-system provides the means for applications to access remote files. This means that a user running a remote X-Window application is able to use that application to manipulate local data.

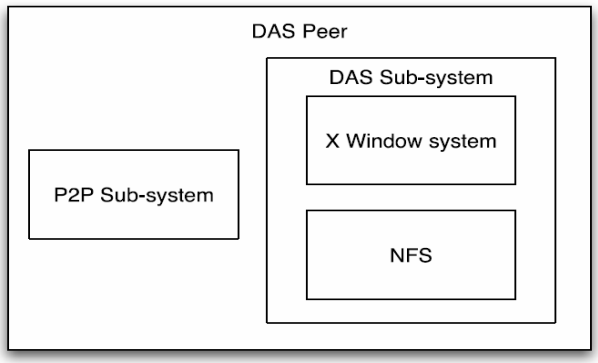

Fig. 2 CPCC prototype implementation

The combination of these technologies provide the foundation needed for remote execution of applications, it therefore makes sense to build the CPCC on top of the Linux platform. The prototype CPCC system can be deployed on any Unix system which provides the X Window and the NFS services.

The prototype CPCC system is built as a standalone application which can be launched by the user. The CPCC application consists of the core Peer and the shell interface which allows the user to access the peer functions.

The implementation of the core Peer is broken down into three components. The remote execution component which uses the X Window system, the remote data provider component which is built on top of NFS, and the custom P2P network component used to connect the CPCC peers. Figure 2 shows the implemented architecture. The remote execution component spawns applications and sets up their IO so that the applications are able to communicate with the remote X Server. The execution of the applications is monitored by the

execution component and the application status is updated when it starts and exits.

The NFS component is invoked by the execution component in order to allow the remote application access to the local user files. Since there is no readily available P2P platform designed to allow remote application sharing, the P2P component of the CPCC implementation has been implemented to address that need.

A. Load Balancing

When copies of the same application are provided by multiple application providers, it becomes necessary for the client peer to have a method for selecting the peer which will execute it. Ideally the application should be run by the peer which will provide the best possible performance.

The performance of the application is governed by the load of the peer which will be executing the application and the quality of the network connection between it and the client peer. When a server is under heavy load the user will experience a slowdown in the application performance. Conversely if the quality of the network connection is inadequate then the application is unable to update its remote display in real-time.

The quality of the network connection is in turn governed by the available bandwidth and the latency. The bandwidth dictates how quickly the data can be sent between the peers, while the latency represents the delay between the time that the data is sent and the time it is received. The client peer must attempt to select the best possible combination of the peer load, bandwidth, and latency when choosing which peer will execute the application. This process is called load balancing.

The client component of a CPCC peer provides load balancing by attempting to choose the server peers which have the optimal combination of load and latency, when requesting applications. When a server peer has too many clients then it will result in undesirable performance for all the clients which are connected to that peer. On the other hand, if the connection between the client peer and the server peer has too much latency then it is a poor choice since it would be unresponsive.

Figure 3 shows how load and latency information is processed in the CPCC implementation. Each peer runs a thread which checks its load at specified intervals and collects load information about other peers. When the thread sends an update request to another peer the time is recorded. The responding peer returns that time and its load. The time is then subtracted from the current time to get the roundtrip time. The average of the round trip time and the load is stored as the rating of that peer. When the user chooses to run an application, the peer selects all the peers which are sharing the application from its list of known peers. The peer with the best rating is then selected as the application provider.

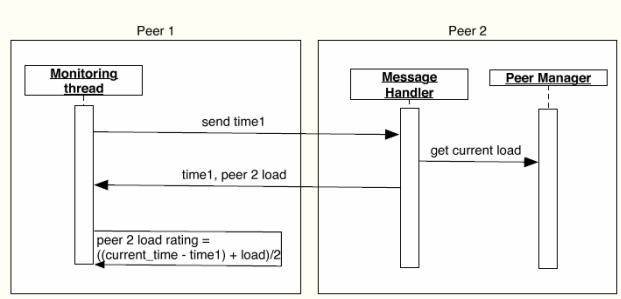

Fig. 3 CPCC load balancing mechanism

V. Experiments of Community-based personal Cloud computing prototype implementation

We conducted several experiments for the CPCC prototype implementation. The experiments were chosen to provide an overview of both functional requirements such as creation of a P2P network and propagation of applications as well as non-functional requirements such as remote application performance.

The objective of the first experiment is to see if a new peer is able to join an existing network consisting of at least one active peer. To join the network a new peer must notify the active peers of the applications it is sharing and retrieve the list applications shared by the active peers. To test that, we start each of the peers in order. When each peer connects to the network we check if the applications shared by that peer are available on the other peers and vice versa.

The second experiment is to add and remove an application on a peer and see if the other active peers become aware of these changes. To test this scenario the user selects and removes an application in the list of shared applications on a peer. When this is done the list of available applications is checked on the other two peers to see if the application is not in the list of available applications. The peer selection algorithm must be tested to see if the peers are selected based on their performance when multiple peers are sharing the same application. In order to conduct this test two peers are loaded with the same application and one of the peers is made to perform intensive calculations such as applying a transformation to a large image or copy large amounts of data. In such a scenario the third peer which is requesting the application must select the peer which is not under heavy load.

The third experiment is to execute an application shared by a remote peer and check if it is able to access the local data. This is tested by running a remote text editor shared by another peer and opening a locally available text file in the editor. Once the application is executed the non-functional testing can be performed.

We used the following application programs to conduct the experiments: GIMP, Kate, and Firefox. GIMP is an image editor. The application was used to load large images. This task requires sending large amounts of data between the peers, which allows us to experiment the performance of accessing remote files. The image must be displayed on the user screen, which tests the sending of complex graphics data. GIMP

allows applying transformations to the image such as blurring and scaling of the image. These transformations are CPU intensive, when such an operation is executed it creates high load on the peer. The Kate application is a text editor. This application allows for testing interactive input by the user. The user is able to open and edit text files with this application. Firefox web browser offers tests of displaying mixed data. The browser displays text and raster graphics data from web pages. The X Window system is optimized for drawing the GUI widgets remotely as well as transferring text data between the application and the remote client. However it cannot easily optimize the transfer of graphics. It is therefore expected that the applications will require large amounts of bandwidth to transfer the graphics data. This will affect the GIMP and Firefox applications. The **GIMP** must send the image document across the network and any time changes are applied to the image a new copy of the image must be sent. It is expected that this will require a large amount of bandwidth, especially when dealing with large images. The Firefox browser needs to send images from web pages to the client peer. This again may require increased bandwidth. In contrast the Kate editor is expected to be more consistent in its responsiveness since it never needs to transfer graphics across the network.

The applications were executed under three scenarios. In the first scenario the load on the peers was low and there was no conflicting network traffic. The results in this case were expected to be good. It was expected the application to be responsive to the user and be able to handle the remote data without noticeable lag. In the second scenario the bandwidth on the network were tied up in transferring large data files unrelated to the CPCC network. It is expected that the applications not to be responsive under these conditions, the user might experience lag between input and the updates in the application interface. When the remote applications must access local files it is expected that there will be lag while the contents of the file are transferred to the application over NFS. The third scenario involved running an application from a peer which was under high load from computational tasks. It was expected that the application to exhibit unresponsive behavior similar to that caused by decreased network bandwidth. The user might experience lack of the interface updates on the application or the application may become completely unresponsive.

The results of the control test are shown in Table 1 which shows that the applications behaved in the same fashion when being run locally and remotely. The notable differences in behavior were in the different path to the user home directory between local and remote applications.

When full100Mb/s bandwidth is available for remote execution there is no perceivable difference between local and remote applications. The screens of the remote applications update in real time when the user interacts with the application.

TABLE I CONTROL TEST RESULTS

Application	Local Time	Remote Time (Seconds)
GIMP	5	7
Kate	53	53
Firefox	1	2

The results of the low bandwidth test are shown in Table 2. As expected the application performance drops as less bandwidth becomes available. When the available bandwidth falls below the 10Kb/s threshold the applications stop being responsive in real time. In other words there is noticeable lag between the user input and the application interface updates. Table 2 shows that the times of the tests performed in the remote applications are slightly greater than those of local applications. The GIMP test was the most affected as the large image had to be transferred over the network after the transformation. The Kate test was not perceivably affected, since the input can be sent to the application faster than the user is able to type it. The Firefox test was not significantly affected by the reduced bandwidth, since the web page data can still be transferred under 3 seconds. This amount of lag is not noticeable to the user.

TABLE II HIGH BANDWIDTH TEST RESULTS

Application	Local Time	Remote Time (Seconds)
GIMP	5	13
Kate	53	56
Firefox	1	2

The results of the high load test are shown in Table 3. Again as in the previous test the application responsiveness drops off as the peer load increases. In the case of high peer load the application performance is the worst of the three scenarios. While there can be observed noticeable lag when the bandwidth is insufficient, the applications can stop responding completely when the peer load is high. It shows that the times of the tests performed in the remote applications were in fact greater than those of local applications. The GIMP and Kate applications were most affected by the high load, while the Firefox application had much lower impact. This test clearly shows that the load of the peer executing the applications impacts the performance the greatest.

As has been observed during the experiments, the application performance is influenced by the available bandwidth and the load of the participating peers. The load of the server peer is the greatest factor in the performance of the remote applications in these tests. The applications are still responsive even when high amounts of bandwidth are being used for other networking. The limitations of the CPCC prototype implementation are not exclusive to the CPCC system, but exist in any remote execution scenario.

VI. Conclusion

In this paper, we propose community-based personal cloud computing as a new model of cloud computing. As such it introduces several new concepts to the fields of cloud computing technology. The paper presents the concept of community application sharing and provides a method for allowing members of a distributed community to share their applications with other members of the community. The shared applications do not need to be copied to and installed on the client machines in order to be used. Since the applications are executed by the application provider, CPCC is potentially platform independent. Members of the community can use different platforms and operating systems, yet still be able to use applications shared by other members of the community.

While P2P networks have been used for sharing files and for communication, no networks currently exist which facilitate the remote sharing of applications. By joining a community and sharing their applications with each other, the users of the CPCC network create a community desktop. The P2P network which backs CPCC does not facilitate file-sharing as typical P2P networks do, but instead it tracks the applications available on the network, and the peers which share these applications. The network provides load balancing to ensure that the applications are executed on the optimal peers.

The CPCC architecture is designed using a modular architecture. The architecture consists of three major subsystems, which include a remote execution subsystem, a P2P network, and a file service. Each of these subsystems is independent of the other subsystem. This allows for an implementation where each of these subsystems is provided by a standalone application. In the example of the prototype, the remote execution is provided by the X Window system. This subsystem can be substituted for a different remote execution system without the need to change the other components.

Community-based personal cloud computing as a new form of cloud computing provides facility for end-user to end-user or personal cloud computing, without depending on applications and services providers on enterprise scale servers.

References

[1] David Chappel. A short introduction to cloud platforms: an enterprise-oriented view. DavidChappell & Associates, Aug. 2008.
[2] Brian Hayes. Cloud computing. Communication of ACM, Vol. 51 No. 7. July 2008.
[3] Tim Jones. Cloud computing with Linux. IBM, Sep. 2008.
[4] Constantine Mantratzis and Mehmet Orgun. Towards a peer2peer world-wide-web for the broadband-enabled user community. *Proceedings of the 2004 ACM workshop on Next-generation residential broadband challenges*, pages 42–49, 2004.
[5] Luciano Paccagnella. Getting the seats of your pants dirty: Strategies for ethnographic research on virtual communities. http://jcmc.indiana.edu/vol3/issue1/paccagnella.html, June 1997.
[6] Richard Quinn. Peer-to-peer networks. http://www.richard-quinn.com/quinn-pages/essays/p2p/peer-to-peer.html, March 2004 (accessed).
[7] Calvin J. Ribbens, Dennis Kafura, Amit Karnik, and Markus Lorch. The Virginia tech computational grid: A research agenda. Technical report, Department of Computer Science Virginia Polytechnic Institute & State University, 2002.
[8] Christian Wagner and Narasimha Bolloju. Supporting knowledge management in organizations with conversational technologies: Discussion forums, weblogs, and wikis. http://wagnernet.com/tiki, June 2005 (accessed).
[9] Lizhe Wang, Jie Tao, Marcel Kunze. Scientific cloud computing: early definition and experience. Proceedings of the 10th IEEE international conference on high performance computing and communication. IEEE press, 2008.
[10] Klaus H. Wolf, Konrad Froitzheim, and Peter Schulthess. Multimedia application sharing in a heterogeneous environment. *ACM Multimedia 95 – Electronic Proceedings*, pages 57–64, November 1995.