REAL-TIME DISTRIBUTED COMPUTING: CHOPS

Sean Lawlor, Patrick Diez, and Frank Ferrie

McGill University
Centre for Intelligent Machines
Montreal, Quebec

ABSTRACT

Recently there has been an emergence of large, indefinite length data (video) streams as used in many computer vision problems. There is now a requirement to transmit these streams between remote locations where post-processing may occur and the results of this may be released to further remote locations for additional post-processing. Using two main networking theories, Chord and the Publish/Subscribe paradigm, ChoPS (**Cho**rd based **P**ublish/**S**ubscribe) was conceived. ChoPS uses a distributed hash table, notably Chord, which is demonstrated to have an average of $O(\log_2(n))$ time in order to disseminate data to n hosts over a unicast network such as the Internet. ChoPS also implements the Publish/Subscribe paradigm, which is inherent in the anonymous video stream subscription application. Along with results demonstrating the efficiency of Chord when used for such data distribution, a full system layout and specification is provided for an implementation of ChoPS.

***Keywords*-Multicast Protocols; Multicast Communication; Peer to peer computing; IP Networks; Computer Network Management**

I. INTRODUCTION

As the Internet continues to expand as a collaborative device, the constraint of unicast communication (in which data have a single destination) becomes of ever-increasing cost. As per RFC 2770 [1] , the IPv4 address space 232.0.0.0/8 is reserved for IP multicast (in which packets have multiple destinations), however, as with IPv6, lack of support and availability of these addresses from consumer ISPs and internet hubs makes this feature unreasonable as a basis for internet-based multicast networking. Instead, as is proposed here, the use of a distributed hash table, notably Chord, provides a means of establishing a virtual multicast network with commodity unicast internet connections that compares favorably against the naïve process of unicasting to each client. Extending Chord, a Publish/Subscribe system and administration interface is implemented to allow seamless collaboration between remote processes.

II. CHORD AND PUBLISH/SUBSCRIBE

II-A. Background

For the purpose of this implementation, it is assumed that a given Internet host's (i's) ability to transmit and receive data is well-represented by two values: its transmission and receive costs, $c_{tx}(i)$ and $c_{rx}(i)$, which may be considered to have units of time, and encompass both the latency (delay incurred by routing) and transmission period (delay incurred by data rate, inversely proportional to bandwidth). The primary implication of this assumption is that transmission cost of a *fixed-length* packet from any host *i* to any other host *j* is constant, and thus that the transmission cost from *i* to *j* is:

$$c_{tx/rx}(i,j) = c_{tx}(i) + c_{rx}(j) \tag{1}$$

The aforementioned naïve process of unicasting to n clients therefore has a total cost of:

$$\begin{aligned} C_{naive}(i) &= \sum_{j=1}^{n} c_{tx/rx}(i,j) \\ &= n \times c_{tx}(i) + \sum_{j=1}^{n} c_{rx}(j) \\ &= O(n \times c_{tx}(i)) \end{aligned} \tag{2}$$

From equation (2), it is evident that the cost of multicasting from i is proportional to the transmission cost for i and the number of clients in the unicast network. In large-scale commercial applications, such as YouTube, the cost of this relationship is favored over the cost of requiring that clients use custom software for multicast communications. However, acquisition of high-bandwidth links for finite-term and small-scale projects may be prohibitive, in which case efficient use of lower-bandwidth links is preferred.

II-B. Chord

Chord is a distributed hash table (DHT) developed at MIT by "*Stoica et al.* [2]" for the purpose of storing data across a large network of *n* hosts, in which lookups, stores, the addition of a new host, and the removal of a host each require $O(\log_2(n))$ time. The core of Chord is a consistent hash function, $h(k) = k'$, which maps host *keys* (called *identifiers*) and data *keys* to hashes. Hashes are organized into a ring of 2^m values, where m is the size (in bits) of a hash, and must be significantly larger than $\log_2(n)$ to avoid hash collisions. The successor of a key k is then the host (or *node*) $successor(h(k))$ such that $h(successor(h(k))) > h(k)$ for the smallest possible value of $h(successor(h(k)))$ if such an identifier exists, and the node with the smallest identifier hash otherwise. This establishes the *ring* of hash values, called the *Chord ring*, in which the successor of a key is the node with the next smallest identifier hash value, and the successor of the key with the largest hash value is the node with the smallest identifier hash value. A key k is stored on node $successor(h(k))$ thus distributing the data stored in the hash table evenly amongst the nodes in the ring, assuming a sufficiently random hash function [2].

Minimally, to perform a lookup, each node i must store the IP address of $successor(h(i))$. Then, if a lookup is performed for key k, the node performing the lookup checks if it stores k. If it does not, it performs a remote procedure call and requests that $successor(h(i))$ lookup k. Given that such remote procedure calls may, in the worst case, require a traversal of the entire ring, this process is $O(n)$ in time complexity [2].

To mitigate this lookup cost, Chord implements a finger table on each node. Instead of storing only $successor(h(i))$, each node i stores up to m finger nodes' IP addresses in a table, where:

$$finger(i,j) = successor(h(i) + 2^{m-j-1}), 0 \leq j < m \tag{3}$$

Clearly, depending on the value of n, not all fingers need to be stored. As is shown in (3), the last finger that needs to be stored

is that for which the following condition applies:

$$\begin{aligned} finger(i,j) &= successor(h(i)+2^{m-j-1}) \\ &= successor(h(i)) \end{aligned} \quad (4)$$

Again, for a sufficiently random distribution of keys, this reduces the size of the finger table from $O(m)$ to $O(\log_2(n))$. Then, to perform a lookup for k, the node i performing the lookup first checks if it stores k. If it does not, it requests that $finger(i,j)$ performs the lookup, where j satisfies the condition:

$$h(finger(i,j)) > h(k) > h(finger(i,j+1)) \quad (5)$$

In equation (5), the greater-than relationship is evaluated while taking into consideration the nature of the Chord ring. Since at each step of the lookup, the use of the finger table reduces the range of hosts left to query by a factor of two, the worst-case time for a lookup is $O(\log_2(n))$ [2].

Details regarding the processes involved in inserting and removing hosts from the ring are not covered here, as the rest of this section is dedicated to an analysis of a proposed modification for Chord allowing multicasting.

II-C. Multicasting

If a given node *i* were to use the same lookup mechanism as Chord to transmit data to a node *j* instead of requesting it, the transmission time would be, logically, $O(\log_2(n))$. Naïvely, a multicast from *i* to the entire Chord ring would require $O(n\log_2(n))$ time, an increase of $\log_2(n)$ over the naïve unicasting process presented in the introduction. However, during this multicast, $finger(i,j)$ would receive (redundantly) the same data to be multicast 2^{m-j-1} times. Thus, optimally, transmitting the packet once to each $finger(i,j)$ would suffice. Each $finger(i,j)$ would then be required to transmit the packet to each $finger(j,j+k)$ for all $k>0$, each $finger(j,j+k)$ to each $finger(j+k,j+k+l)$ for all $l>0$, and so on, as shown in Figure 1.

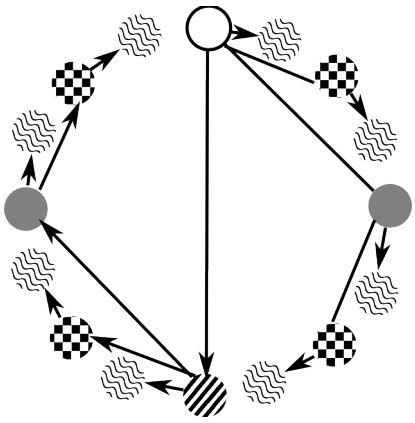

Fig. 1. Chord multicasting paths using 16 nodes

Assuming all transmission times between nodes are identical, this process is $O(\log_2(n))$ by virtue of the fact that at each time step, the number of nodes having received the multicast data doubles. Additionally, this inherently provides the basis for the publish/subscribe system, as all hosts participating in a Chord ring will have to retransmit data multicast to the ring, irrespective of their need of it. In this sense, the Chord network itself acts as the broker (the host in publish/subscribe normally responsible for matching data producers with data recipients).

Stoica et al. [2] focus on the cost to *lookups* of network latency, whose analysis cannot be assumed to be reflected in the process of multicasting. For the purpose of analyzing multicasting, we recast the multicasting process as a recursively defined cost function:

$$C_{multicast}(i,j) = c_{tx/rx}(i, finger(i,j)) + \begin{cases} 0 & \text{for } finger(i,j) = finger(i,j+1) \\ max(C_{multicast}(i,j+1), C_{multicast}(finger(i,j),j+1)) & \text{otherwise} \end{cases} \quad (6)$$

where $C_{multicast}(i,j)$ is the cost of multicasting from node *i* to all nodes $finger(i,k)$ for $k \geq j$. Assuming $c_{tx/rx}(i,j)$ is not constant, there exists at least one ordering of the nodes in the Chord ring that minimizes $C_{multicast}(0,0)$, the Chord multicasting cost for node 0. Analysis of the directed graph of the Chord ring, in which vertices are nodes and edges are weighted with transmission costs is difficult, and expected (though not proven) to be at least NP-hard by the following argument:

> Since nodes are only capable of multicasting after having received the data to multicast, maximizing the initial slope of the function of the number of transmitting nodes vs. time will in some sense minimize the multicast time. In turn, maximizing this slope implies that the first nodes to transmit ought to have the least transmission costs to their fingers of any nodes in the ring. Conversely, the last nodes to transmit ought to have the greatest transmission cost. However, these nodes will be transmitting to their successor along the "outside" (see Figure 1) of the Chord ring. As maximizing the edge weights along the outside of the Chord ring is equivalent to the traveling salesman problem, a known NP-hard problem [3], this suggests that the problem of minimizing the Chord multicasting cost for a given set of nodes (by moving nodes in the ring) is also an NP-hard problem.

Consequently, determining optimal-case and worst-case for a given set of nodes is an unfeasible approach to evaluating the performance of Chord multicasting, if the number of nodes is not impractically small.

II-D. Results

Two sets of simulations were performed. For the purpose of simulation, the Chord ring was simplified to ignore the hash function, and instead consists only of evenly spaced nodes. This simplification is considered to have a negligible effect on the results, given that the number of nodes is small with respect to the number of possible hash values. A variation of an exponential distribution was used to generate random host transmit and reception costs in the interval $(0.1, 1.0)$. Additionally, the expectation value of the distribution is specified to mimic the uneven distribution of high-speed and low-speed consumer connections on the Internet. Each result was obtained from 1000 simulations performed with different randomly-distributed transmit and reception costs.

Figure 2 shows the time required to multicast a single packet with respect to the number of nodes. For these simulations, the expected transmit and receive costs were 0.7 respectively. Clearly, in 95% of all cases, the multicast cost increases logarithmically with respect to the number of nodes in the ring. The second simulations, whose results are in Figure 3, measured multicast cost with respect to expected transmit and receive cost for 256 nodes. If all nodes were identical, these results would be perfectly linear, since the Chord ring topology is optimal and invariant in these simulations. However, these data have negative inflection, suggesting that multicasting is negatively impacted by the variance of the transmit and receive costs in the ring (which is largest when the expected cost is the average of the bounds of the distribution, making the latter uniform).

It is concluded from these data that Chord is applicable to networks with varying point-to-point communication costs, and thus is suitable for use for multicasting on the Internet.

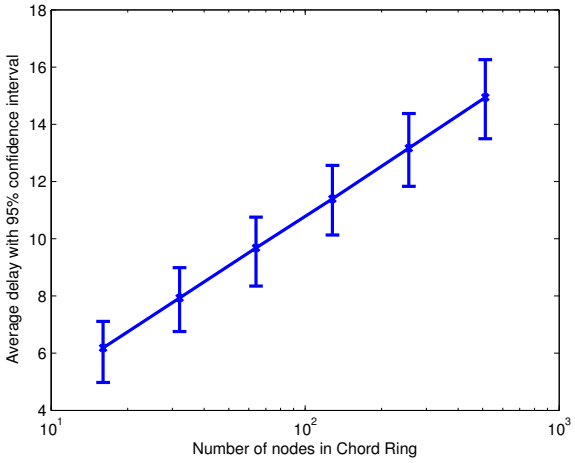

Fig. 2. Graph of Multicast Cost vs. Number of Chord Nodes (logarithmic) with a Transmit/Receive Cost Expectation Value of 0.7

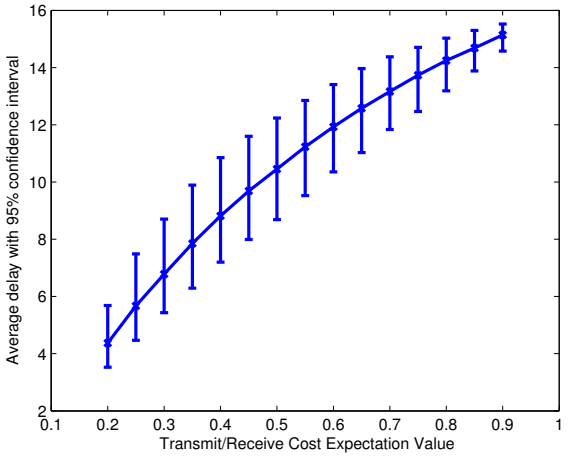

Fig. 3. Graph of Multicast Cost vs. Transmit/Receive Cost Expectation Value with 256 Participant Nodes

III. IMPLEMENTATION WITH CHOPS

III-A. ChoPS

ChoPS is the name given to an implementation of the Chord protocol combined with the Publish/Subscribe protocol as outlined in this paper. It defines an implementation which employs both concepts to create a distributed real-time system which allows for indefinite length data to be transmitted without knowledge of receivers (subscribers) nor the network transmission protocol. ChoPS will also allow for data multicasting to be optimized, providing that each client within a ChoPS network (*projects*), maintains the ability to retransmit the data it has received to other computers within a project (*nodes*).

ChoPS allows for a simple front-end to large, complex data transmissions (i.e. *streams*) to a largely distributed network through four features: (1) Use of Chord for multicast data transmission. (2) Use of the Publish/Subscribe paradigm to allow for a loosely coupled network. (3) A simple server which manages all connections. (4) A graphical user interface (GUI) to manage the entire system remotely (administrative frontend).

III-B. Use of Chord and Publish/Subscribe in ChoPS

As previously outlined, Chord allows for data stream transmission to be optimized when sending data to a large number of nodes. By using the Chord mechanisms we can create projects within ChoPS. These projects allow for multiple nodes to communicate with each other while remaining separated from other projects. This guarantees that a node only contributes to stream transmissions of the projects of which it is a member.

A ChoPS project is a Chord ring where the published stream within that ring is of a defined topic as outlined in the Publish/Subscribe paradigm [4] . This design of building a Chord ring for each project allows for projects to be separated from each other, and guarantees that each node within a project will have its maximum bandwidth to (re)transmit the stream it is receiving to the other nodes within the project.

The data transmitted across a ChoPS project is of indefinite length and format. It therefore could be of infinite length, such as a security video stream, which never ends as long as the camera is active. Granted a video stream is only one of a near-infinite number of data types which can be transmitted with ChoPS, as ChoPS is designed for generalized data stream transmission.

III-C. ChoPS Server

The ChoPS server is implemented as a system daemon, which opens port(s) on a node, each of which can accept connections. The server can handle multiple concurrent connections from multiple administrative clients and/or project nodes. Each connection has its own packet listener, which simply sleeps until a packet is received, and then passes the packet to a centralized packet handler which processes the packet information and sends a response accordingly. Each client connection, whether from an administrative client or simply a stream transmission through a project will get its own connection information which the server is aware and can process independently from other client connections.

Each node within the ChoPS network runs a local instance of the ChoPS server. The server initially sleeps, only listening for administrative connections until an administrator tells the node to connect to a project and start (re)transmission of a stream. Each node stores all the projects it is connected to, as well as information about each project which is required to participate in stream transmission within the project (finger-table, successor, etc.).

III-D. Administration

An administrator can log into the server through the GUI and have two administrative roles. The first role is to administer a single node, either shutting it down, starting a new project, joining a project, or removing itself from a project. The second role is to administer an entire project, either shutting down all participating nodes (for that project only) or modifying some project setting such as the administrative credentials. Each of these roles are separated from each other for system stability when managing ChoPS.

III-E. Project Monitoring

A user can also log into a ChoPS project through a more restricted account to simply get information about a project. They cannot change any aspects of the running project nor are they able to change any aspects of any node in the project. This mode is designed for unprivileged users to be able to gather information about the network and if necessary notify the administrator of a change that needs to take place.

III-F. Server Data Processing

Once a node is a member of a project, and the project has been notified that a new member node has been added, the node starts receiving the data which is being published across that project. The node may direct the data it received in one of two ways: it may drop the data and simply participate in retransmission or may subscribe to the data stream and begin processing it. Once the data has been processed, the node also has the option to publish the data into a new ChoPS project for other nodes to connect to and further process/view/retransmit.

If the node is receiving the data and passing it into the userspace (the computer system area where a users' programs run in user-editable memory), the node will create a local socket to which the stream is passed and a system user can then connect to the stream and process the incoming data as it is received. For example, the user could connect to the local socket, process the data using a tool such as Matlab, and then send the output to another stream which can either be viewed locally or passed back to the ChoPS server to become another publication in a new ChoPS project. However, even if a user is processing data, should the node not be a data endpoint in the Chord process, the node will still retransmit the data to the nodes listed in its finger table. All nodes must participate in the ChoPS data transmission protocol regardless of whether they are reading and/or storing the stream locally.

III-G. Test System

A test system of ChoPS has been implemented currently transmitting video within a closed network. As suggested in the original Chord [2] paper, an SHA hash is utilized in this system to generate node identifiers. This implementation was created in Java™, and is fully cross-platform capable, allowing for many different system types to be able to generate, process, and review data transmitted through ChoPS. In this system, data was captured from a simple laptop capturing JPEG frames from a webcam which was then transmitted through a ChoPS ring consisting of variations of 4, 8, and 12 hosts (the last being specifically not a power of two for the purpose of analyzing unbalanced Chord rings). A Canny edge detector [5] was then performed and the resulting data stream published into a small (2-node) ring which was then displayed, alongside the original data stream.

Throughout all trials, ChoPS never exceeded 2% CPU utilization of any single system (with processors no less than a single 1.7 GHz code) and never exceeded 10 MiB of memory usage on any system. These measurements were done on a video stream producing 20 frames per second with each frame being 16 KiB in size. None of load these measurements include the edge detector or tools used to capture and display the videos.

IV. FUTURE WORK

There are many applications for ChoPS in current research. For example, it could be very helpful for multiple universities participating in a research project where data is constantly being generated and processed across a large physical distance requiring transmission over the Internet.

Future work can be directed along a series of different avenues, the first being applications other than video processing (as was done with the Geoide project). Since ChoPS is not specific to what types of data are being processed, it is not expected that ChoPS will perform incorrectly, however it should be investigated to confirm this theory.

Another avenue of future work would be to add auto-detection and -correction of ring stability to ensure no nodes die unexpectedly without a ring rebalancing to take place. However, this feature would also need to make sure that bandwidth is not greatly reduced for standard data transmission when adding the stability detection message transmissions.

Finally, further testing should be performed for a full empirical analysis of our time bound in real networks. So far, testing has only been performed on closed, private networks and therefore other traffic's influence on ChoPS needs to be investigated.

V. CONCLUSION

There are many options to distributed computation, however, many current systems do not take into account the possibility of transmission of large, possibly infinite, data streams in a multicast nature to many end hosts. Also the current implementation of the IPv4 internet backbone does not allow for a simple multicasting protocol to be implemented. To build a multicast network across a large, distributed collection of hosts, there currently exists two main options: to spend large sums of money and setup a dedicated routing network which would allow the multicast protocol, or to tunnel all traffic over a dedicated channel and increase network overhead. By using the Chord DHT model coupled with a Publish/Subscribe protocol, ChoPS can mitigate these problems by providing a stable and efficient environment for these data transmissions.

The features outlined within the ChoPS administrative graphical user interface allow for basic users to start right away with the ChoPS system and begin processing their data. By leveraging Chord's proven stability, ChoPS guarantees that data transmission will be consistent throughout a project's life. ChoPS employs practices which are standard with Unix based processes, which have been shown to have superior security and stability. For example, all commands to the ChoPS server are outlined as ASCII text, allowing any system administrator to login without the GUI and control any aspect of the system.

Finally due to ChoPS generic nature, the system is not limited to video stream transmission, as was requisite for the Geoide PIV-17 project, the former parent project to ChoPS. ChoPS can be applied to a data stream of any type and nature with a greatly increased number of participants.

ACKNOWLEDGMENT

Thanks to Professor Frank P. Ferrie of *McGill University* and Professor James H. Elder of *York University* for supporting this system in *Geoide PIV-17*.

VI. REFERENCES

[1] D. M. of Cisco Systems and P. L. of Sprint, "Rfc 2770: Glop addressing in 233/8," Network Working Group, Tech. Rep., 2000, http://www.rfc-editor.org/rfc/rfc2770.txt.

[2] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *Proc. IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17 – 32, Feb 2003.

[3] R. M. Karp, *Complexity of Computer Computations - Reducibility Among Combinatorial Problems*. New York, New York: Plenum, 1972, pp. 85–103.

[4] T. J. K. Birman, "Exploiting virtual synchrony in distributed systems," in *Proc. of the eleventh ACM Symposium on Operating systems principles*, 1987, pp. 123–138.

[5] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 6, pp. 679 –698, Nov. 1986.