Hypercube: a Peer-to-Peer Network

Hardeep Singh, 3089450 Manjot Singh, 3087643

Abstract— **The popularity of the peer-to-peer networks is highly increased over the years. It becomes very useful medium to store and share large amounts of data. To simulate a structured peer-to-peer system, we have used hypercube topology to create network. Moreover, we have stored data on various nodes present in the network. The paper is also highly focused on the way how data is retrieve from the nodes present in the structured peer-to-peer systems. Apart from this, reliability of the system is also tested by killing some of the threads from the simulated peer-to-peer system.**

I. INTRODUCTION

Peer-to-peer networking is a distributed computing application architecture that divides the workloads between peers[1]. Where peer is referred to a single workstation present in the network. Peer-to-peer systems have become a popular area of research. The reason behind its popularity is that network can store and share large amount of data among peers. The most popular peer-to-peer systems nowadays are Bit torrent, Freenet, Skype, Gnutella etc. In the distributing computing, versatility and scalability are the key factors. The performance of distributed systems always measures on the basis that how well data is distributed over the network and how large a network can be. It further leads to the concept of network topologies present in the peer-to-peer networks. Hypercube network topology is one of the most versatile, scalable and most efficient network topology known till date. The reason behind it is its regular structure and efficient algorithms.

In this paper, we have used Hypercube network topology to simulate our peer-to-peer system. Using hypercube, we have created structured network and then used it for data storage and data retreivel purposes. We have used a distributed algorithm to create a peer-to-peer system. Moreover, after creating network using hypercube topology, we have stored data on the various nodes present in the system. In addition, we have used an algorithm for data lookup at nodes of the developed peer-2-peer system. At last, we have checked how reliable our system is and how it self-organized when a node gets killed from the network.

In section II, we have discussed about the related work. The methodology and the algorithms used in this paper are discussed in section III. In section IV, we have write about the experiments performed and the results, followed by discussion and conclusion.

II. RELATED WORK

The peer-to-peer systems are the well-researched field of the computing world. There have been various papers written on peer-to-peer systems using different network topologies. Mario Schlosser, Michael Sintek[2] used a graph topology called Hypercube to create a peer-to-peer system. They demonstrates a routing algorithm for hypercube topology, which results in very efficient for broadcasting and searching in the topology. Moreover, their system was self-organizing and resilient against failure. In our system, we created a simulated hypercube topology, which shows how data can be stored on the nodes of system and how stored data can be retrieved.

Ali Ghodsi[3] provides an algorithm to broadcast a message to all nodes present in a given structured overlay network. The messages sent to the nodes are all unique, with no redundant messages. Apart from this, they provide a case study for the implementation of Application Level Multicast on structured overlay network. However, in our paper we focussed on storing and retrieving data from the nodes in the given network. Moreover, we tried to kill some of the nodes from the network, and shows whether network self-organizes or not.

Andi Toce, Akira Kawaguchi[4] created a labelling scheme which assigns identifiers or labels to each node and use that identifiers to determine the distance between one node to every other nodes in the system. These labels allows them to create a virtual overlay network very similar to the hypercube network. They prove that the labelling scheme allow to reduce the bandwidth utilization in the network. However, In our paper we are using actual hypercube topology to create a peer-to-peer system. Moreover, while storing data on the nodes, we have calculate the number of hops to send the data from source node to destination node.

Stephanos and Diomidis[5] have done survey of peer-to-peer systems. They have mainly focused on the content distribution systems. They have discussed about Structured, Unstructured and Hybrid peer-to-peer systems. Moreover, they discussed about the scalability problem in the peer-to-peer systems and how it can be increased. There paper is lacking of the practical implementation of the concepts described. However, in our paper we have created a topology called hypercube and tried to store and data from its nodes.

III. METHODOLOGY

Hypercube network topology is one of the most versatile, scalable and efficient network topology. It can be directly used for a structured peer-to-peer architecture. In a n-dimensional hypercube network, we have d number of nodes, where $d = 2^n$.

In a hypercube network each node have its own n-bit ID. Each of its node is connected to n number of other nodes, e.g: the nodes that have IDs different by exactly one bit. Each data item present at the nodes in hypercube network is uniquely associated with a hash key.

Program follows 4 goals or targets –

1. Hypercube is constructed for a peer to peer network.
2. Files are created and stored on nodes.
3. Retrieve files from nodes.
4. Ensuring reliability of network.

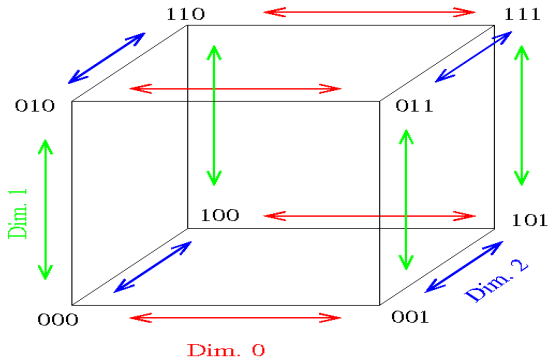


Fig 1. Hypercube [6]

Goal 1:

To perform goal 1, we created a class P2PNode, where each thread represents a separate node. n defines number of hypercube dimension. We use helper function init_nodes() to initialize our structure where we create a list of 2^n nodes with n connections forming a hypercube. We use binary identifier to represent to each node. P2PNode class contains constructors and several functions that upon start of each thread attempts to find neighbors of the node (neighbors are nodes whose id differ by 1 bit). After node is created and its peers are found, nodes are able to send and retrieve messages using class queue (imported from python library). We use command line arguments to input n, so imported module argsparse from library which makes it easier to write such interfaces. We define several arguments it requires and it analyzes list of command line instruction provided to python script. We used another module from library i.e. logging so that all python modules can participate. We use module level functions especially debug and other similar functions like info, error. Logging is thread safe and especially used when working in multi-threaded environment.

Goal 2:

We create additional functions to create files, calculate hash and store file in class P2PNode. m is number of files created at each node. k is key which is generated by taking ascii of file name, summing it together, dividing by 2^n and getting its remainder. Bit distance is used to determine the path by which files are passed.

File can be stored at node that generated it or passed through network to store at another node whose file hash matches with node id. Node compares key with destination id to see how many bits match and pass the file to node with most matching bits. Track is list of nodes that message passed on its way to reach its destination. If track has 1 element, it means file is stored on same node where it was created. Function do_other_things is used to simulate user doing other things. We also calculate average number of hops required to store file in network to find its efficiency.

Algorithm for producing files:

```
files_produced = 0
while files_produced < 'm' {
        work random time between 1 and 10
        produce random file and key
        store file in node with identifier = key k
        files_produced += 1
        work random time between 1 and 10
}
```

Goal 3:

Now in Class P2PNode, we have added methods to retrieve the data stored at the nodes of peer-to-peer network by hash value. We have added new parameter 'p', which is taking care of the number of files to be retrieved. p can be set a maximum value of m i.e. number of files generated. When a node wants to retrieve a file from another node, it sends "get" message to the node keeping the file. The other node will then sends the "get" message back to sender with requested file. We retrieve a random file from available files present in network along with its track, given a key. Generate key of file to retrieve as a random retrieve file in node with identifier equal to key. If file is stored on node itself it retrieves file immediately otherwise it sends a message to node where file is stored.

Algorithm for lookup and retrieve data:-

```
File_retrieved = 0
While files_retrieved < 'p' {
```

Work a random time between 1 and 10 seconds
Key_of_file_to_retrieve = produce a random value between 1 and ‘m’.
Retrieve the file in the node with identifier = key_of_file
Files_retrieved = files_retrieved + 1
Work a random time between 1 and 10 seconds
}

Goal 4:

The given n-dimensional hypercube is producing, storing and retrieving data. Now to check reliability of the given structured P2P system, we have introduced two new parameters ‘q’ and ‘r’, where q is the number of nodes to kill and r is the number of seconds. We created a class evil that destroy random nodes in network. From running nodes in the network we pick a random node and it becomes victim. Run function is used to kill that node and its info is logged and number of nodes killed is incremented by 1.

Algorithm to kill threads in system:

Nodes_killed = 0
While nodes_killed < ‘q’ {
Work for ‘r’ seconds
id_of_victim_node = produce a random value between 1 and n
kill the victim, id = id_of_victim_node
}

IV. EXPERIMENTS

Here we are performing the experiments on this network varying different parameters and calculating average number of hops.

1) *Varying parameter n*

In first experiment we take n i.e. number of dimensions in a hypercube and generate 2^n nodes for structured peer to peer network. We also found and printed peers of each node based on bit difference.

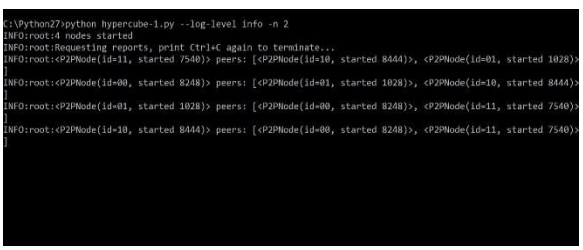


Fig 1 Nodes statistics (n=2)

We also found that for virtual machine maximum value of n can be 11 after which it is unable to create more threads.

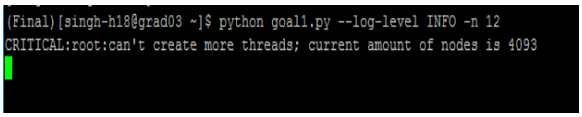


Fig 2 Crashing at n =12

2) *Varying parameter m and n:*

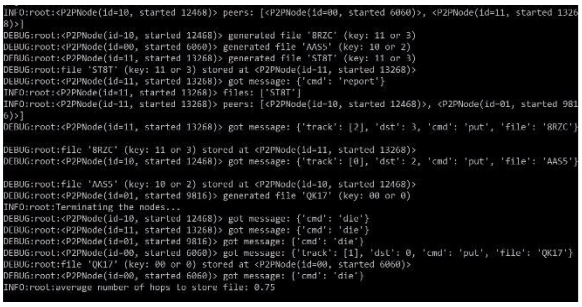
We have already generated hypercube with 2^n nodes. Now we store files on each node based on parameter m(determines number of files generated at each node). We generate encrypted files as four character strings that contains uppercase letters and integers along with key.

The number of files each node can store is only limited to RAM available. It is difficult to determine how large ‘m’ can be because program with combination of n and m would need a very long time to run. We have tried it for some values of m and n:-

	n=2	n = 5	n = 11
m = 2	0.75	2.45	5.38
m = 10	0.70	2.62	6.10
m = 50	1.06	2.24	6.01

Table 1: average routing hops for different n, m values

We then print report to see how files generated are stored on nodes that generated them or on remote nodes based on track. We also print where each file is at termination. Every time node stores a file it also stores its track so we can calculate average number of hops required to store files. By varying m and n we generate different value for number of hops.



s

Fig 3 Showing data at each node

3) *Varying parameter n, m and p:*

We have generated hypercube and stored file on each. Now we want to retrieve that file based on parameter p(determine number of files to be retrieved). P is less than or equal to m. Files are passed along nodes based on bit distance.

	m=1, p=1		m =10, p = 5	
	Store	retrieve	Store	retrieve
n = 2	1.0	2.0	0.87	2.0
n = 5	2.47	4.93	2.49	4.87
n = 11	5.51	11.02	5.49	10.99

Table 2: average routing and retrieving hops for different n, m and p values.

When nodes want to retrieve a file from another node it sends “get” message with file key as “dst” shown in got message string and is shown in output as node retrieve file ‘AB1D’ and other node that send file, sends “get” message back with requested file shown as sent file “AB1D” to querying node. We also show average number of hops to store or retrieve a file.

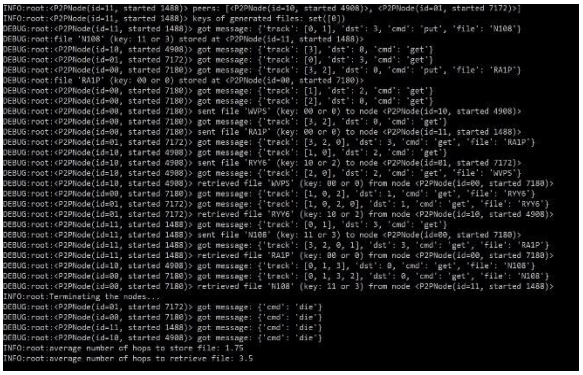


Fig 4 Showing data retrievel

4) *Varying parameter n, m, p, q and r:*

We have successfully generated hypercube, store and retrieved files on and from nodes. We now check reliability of our peer to peer system. We included 2 new parameters q(number of nodes attacked) and r(evil mastermind time).A new class evil selects random node and generate command for it to die.

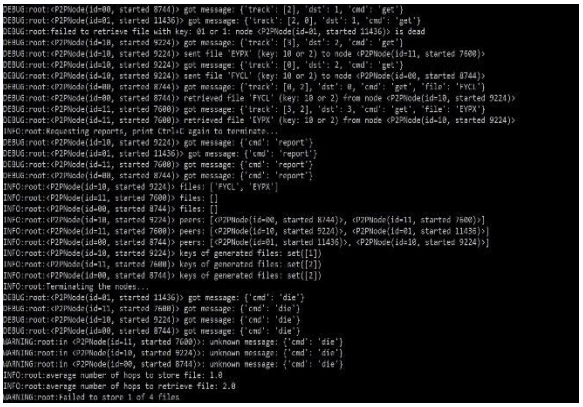


Fig 5 showing killed nodes and the remaining nodes

We show average number of hops required to store and retrieve files when a node dies along with failure of storing and retrieving files when evil mastermind is in action.

V. DISCUSSION

We have created a simulated peer-to-peer structured architecture using hypercube topology, where each node is simulated by using one thread. The virtual machine used for this project can create maximum 2048 simulated nodes. The hypercube topology came out to be very efficient in storing and retrieving data from the nodes. Moreover, it is very reliable.

In future, big companies can use hypercube network for storing their data and retrieving their data. In addition, they can also trust on the reliability of the hypercube peer-to-peer architecture.

VI. CONCLUSION

The project has presented a method for creating structured peer to peer network that is organized as an ‘n’ dimensional hypercube and how we are able to create files that are stored on each node and messages are exchanged between nodes. It also provides capability to look up and retrieve file from node where it is stored in network.

Reliability of network is tested against an evil process that kills random nodes to see its effect on other network activities. Several experiments show how much we can modify each parameter to generate network and functions that virtual machine supports. Average number of hops required for each parameter combination provides context of performance.

REFERENCES

[1] https://en.wikipedia.org/wiki/Hypercube

[2] Mario Schlosser, Michael Sintek, Stefan Decker, Wolfgang Nejdl, “ HyperCup – Hypercubes, ontologies, and Efficient search on Peer-to-Peer Networks”, Stanford University, Stanford, USA, 2003.

[3] Ali Ghodsi, “ Multicast and Bulk Lookup in Structured Overlay Networks”, 2010, Unpublished.

[4] Andi Toce, Abbe Mowshowitz, Akira Kawaguchi, Paul Stone, Patrick Dantressangle, Graham Bent, “ An efficient hypercube labeling schema for dynamic peer-to-peer networks”, 2017.

[5] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, “ A survey of Peer-to-Peer Content Distribution Technologies”, Athens University of Economics and Business, 2004.

[6] Philipp Berndt, Matthias Hovestadt, Odej Kao,” Characterizing latency in periodic P2P hypercube gossiping, 2012