# ComP2P: Application of Distributed Hash Tables to Distributed Computing

Chirag Sangani
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
chirags@iitk.ac.in

*Abstract* - **This paper proposes a protocol for distributed computing called "ComP2P". ComP2P utilizes Distributed Hash Tables (DHTs) at its core to provide a general purpose distributed computing platform. The DHT called "Kademlia" is selected for this purpose since its properties allow for optimization for different computational tasks. The result is a swarm of nodes which combine and share their computational power, with certain nodes or clusters possibly specialising in certain types of computations. The combined computational power can be accessed by any node in the swarm. This paper describes the similarities between file-sharing and distributed computing using DHTs, a modification to the Kademlia protocol to allow for specialized clusters and provides directions for future research.**

Keywords

ComP2P, distributed computing, cloud computing, parallel computing, P2P, peer-to-peer computing.

## I. Introduction

Distributed computing is a branch of computer science that deals with the regulation of a pool of shared computing resources contributed to by computers (hereby called as "members" or "nodes") over a computer network. The nodes in such a collection, collectively known as a swarm, communicate with each other to achieve a common goal set forth to the swarm by some entity within or outside of the swarm. Thus, to a user querying the swarm with a particular task, the entire swarm appears as a single computing entity with capabilities far beyond a single node. The inherent advantages of such a system are cost-effectiveness as compared to a single, high-end computer and reliability as compared to a centralized system due to the absence of a single point of failure [7].

Peer-to-peer (P2P) networking is a type of a distributed computing architecture in which all peers partition tasks equally among themselves. Peers are all of equal status. In contrast to a client-server model in which a client is either always a consumer or a producer of a service, peers in a P2P network are both suppliers as well as consumers of the service. The origins of P2P lie in localised movements of individuals desiring to share files and resources over a network in an honour-based system in which a node produces at least as much as it consumes. P2P systems were popularised by file-sharing services such as Napster and have, of recent past, received considerable attention due to the success of the BitTorrent protocol [19].

There are a number of similarities between the problems faced by the fields of distributed computing and peer-to-peer networking. Both involve the management of resources spread over a network, the maintenance of an overlay structure to facilitate communication, reliability, availability, and fairness. The differences between the two are in the variation in capabilities of nodes and in the nature of the producer-consumer relationship.

While peer-to-peer file sharing networks usually have an equal number of producers and consumers, this ratio is highly skewed in the case of classic distributed computing projects. As an example, we shall study two infrastructures available for initiating a distributed computing project: the Berkeley Open Infrastructure for Network Computing (BOINC), a middleware system developed by a team led by David Anderson at the Space Sciences Laboratory at the University of California, Berkeley; and CompTorrent, a general purpose distributed computing platform that uses techniques derived from the popular BitTorrent file sharing protocol. The former is described as a "quasi-supercomputing" platform with about 484,458 active computers worldwide and an average processing power of 5.605 PFLOP/s as of April 2011 [5]. BOINC is free software for anyone desiring to start a volunteer-driven distributed computing project [6]. Based on a server-client model, a BOINC project relies heavily on the availability of a server to keep the distributed system alive and functional within normal parameters. The capabilities of the server also determine the scale of the volunteer effort supported and, hence, scalability becomes an issue for large projects. The latter, CompTorrent, is a proposed distributed computation system inspired by peer-to-peer networking principles. It relies on a modified version of the BitTorrent protocol to implement distributed peer-to-peer computing [1]. To participate in a project, a volunteer must download a *.torrent* file specific to the project which contains metadata about a tracker. In BitTorrent terminology, a tracker is a server which assists in the communication between peers. In the original protocol, a peer is required to communicate with a tracker to initiate download or upload. The peer is also required to periodically communicate with the tracker to receive information about new peers as well as to provide statistics [4]. While Bit-

Torrent is inherently peer-to-peer in nature, the dependence on a single critical point, namely, the tracker, creates issues with regards to reliability and scalability. The analogy of sharing a particular file also does not convincingly carry over to the distributed computation paradigm, since a node might want to supply computation resources regardless of the project. CompTorrent requires that a node manually "plug-in" to a project by downloading the *.torrent* file.

Historically, most major distributed computing projects have been initiated for scientific purposes. A study of projects developed using the BOINC platform shows that most projects are research projects related to the fields of biology, medicine, earth sciences, mathematics, physics and astronomy [14]. While the need for computational power is understandably significant in these fields, an average user might not be interested in initiating such projects. Such a user might desire to increase the computational power available to him / her for daily tasks such as running productivity suites, rendering images or encoding audio or video files. In such cases, using existing frameworks might prove to be ineffective, cumbersome and impractical.

The answer to this problem has arrived recently in the form of distributed, or cloud computing. Based on the same concept as of a shared pool of computation resources, a client is required to possess only a minimal machine and network access to accept services from the cloud [17]. An example of such a system is the Google Chrome OS, a Linux-based operating system designed to work exclusively with web applications. The interface of the Chrome OS takes a minimalist approach, resembling a web browser in its entirety [15]. The user accomplishes his or her tasks "on the internet" using popular online services for productivity, file storage, etc. This approach, however, has certain issues with regards to availability, reliability and privacy. Since the client is merely a window to the "cloud", a network connection is a must to accomplish any task or to access any data. Since most services are provided by corporations, there is a likelihood of an outage of service affecting millions of users. The year of 2008 saw five major outages within a span of six months for the popular email service "Gmail" [3]. People who used Gmail for business purposes complained about these outages. Another example is the outage of the T-Mobile Sidekick platform, a mobile device that relies on cloud computing for access to calendar, address book services and other key data. An outage of several days left users unable to access the internet or their address books [8]. Finally, there remains an issue of principle, in that participating clients do not add to the pool of computation resources, but only drain it. This makes the paradigm similar to the server-client architecture.

This paper introduces a new system called "ComP2P" which is a decentralised, community-driven, peer-to-peer, cloud computing platform that utilizes distributed hash tables. The author envisions a system where individual home users or universities can participate in a worldwide effort to pool computational power for use by any participating user who desires to do so.

A brief summary of distributed hash tables and Kademlia, the DHT of choice, will precede the introduction to ComP2P for the benefit of the unacquainted reader.

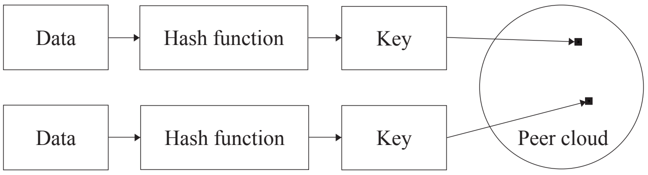

Fig. 1. Key-value map in a DHT. The hash of the data gives the key, which maps to a unique peer in the cloud. This peer holds the data associated with the key.

## II. Distributed Hash Tables

### *A. Overview*

A distributed hash table is a decentralised distributed system which provides a lookup service similar to hash tables [2]. *(key, value)* pairs are stored in a DHT; any participating node can efficiently retrieve the value associated with a given key. The responsibility for maintaining the mapping from keys to values is shared by all the nodes participating in the DHT in such a way that a change in the overlay topology causes minimal disruption. This allows DHTs to be highly scalable, reliable and makes them capable of handling frequent operations of node joins, departures and lookups. DHTs form an infrastructure for more complex services such as file sharing, content distribution, etc. A well-known service which uses DHTs is the BitTorrent distributed tracker.

As shown in Fig. 1, data to be assigned to a node inside a DHT is hashed using a hash function to obtain a key. In Kademlia, the hash function used is SHA-1, and the key size is 160 bits [16]. Peers in the cloud are assigned a unique identifier in the same number space as the keys. This is usually accomplished by applying the hash function on the IP address of the node. Nodes are then assigned those data values, whose keys are "close" to the respective node identifiers, the definition of "closeness" varying between different DHT protocols.

A *lookup* query in a DHT refers to the process of, given the key to some data value, determining the address of the node to which that data is assigned. Lookups are typically of two types: iterative or recursive. The former requires the querying node to negotiate information with its neighbours until the target node is found (or it is determined that the key does not exist in the network). In the latter method, the querying node requests its neighbours to query the network on its behalf. In either case, it is important that the query terminates and does not exist indefinitely if the target node is not found. This guarantee is provided by the lookup algorithm of the DHT protocol.

The advantages of using a DHT as an infrastructure for a service which is distributed in nature are: efficiency of a lookup query, reliability of the overlay network, fair distribution of workload and scalability of the peer cloud [2]. DHT implementations usually do not require a node to possess knowledge of

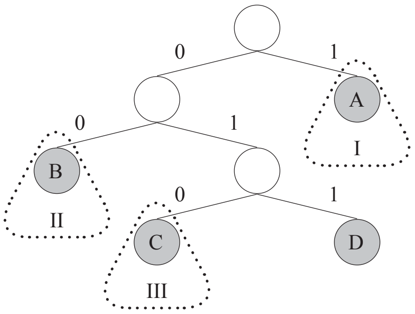

Fig. 2. A Kademlia binary tree with four nodes whose identifiers are: (A) 10000, (B) 00100, (C) 01001 and (D) 01100. The path leading to a node represents the shortest unique prefix of that node. The encircled sub-trees represent the three highest sub-trees corresponding to node D, which do not contain the node D.

the full network; for example, Chord and Kademlia require that each node remembers only $O(\log N)$ nodes, where $N$ is the total number of nodes in the peer cloud. The overlay network is reliable - the threshold value for number of node failures required to affect the stability of the entire network in general is usually large, which makes a global outage highly improbable [20]. Finally, given a large keys pace of 160-bits, a DHT can scale up to easily accommodate from a few hundred to a few hundred thousand users. These advantages make DHTs highly suited for peer-to-peer distributed computing.

### *B. Kademlia*

"Kademlia" is a distributed hash table that uses a novel XOR-based metric topology [16]. It offers a number of features such as a minimal control overhead, parallel and asynchronous queries designed to reduce time-out penalty, a flexible routing mechanism that prefers low-latency paths, and resistance to basic denial of service (DoS) attacks.

Kademlia assigns, like other DHTs, 160-bit keys and identifiers in the same space to data and nodes respectively using the SHA-1 hash of the data or the node's address. The majority of the benefits of Kademlia are derived from its XOR-based metric topology, which is described in the remainder of this section. This metric can also be exploited to optimise Kademlia for peer-to-peer computing (as explained in section IV), thus explaining Kademlia as the DHT of choice for this proposal.

Kademlia arranges the peer cloud in a binary tree-like logical structure in which nodes are at leaf positions. The position of each node is determined by the shortest unique prefix of its identifier. The leading bit of the prefix determines the child of the root node of whose sub-tree the node belongs to. The next bit determines the next sub-tree, and so on, till a sub-tree is found such that the prefix represented by the node is unique in that no other node possesses that prefix. Fig. 2 shows an example tree with four nodes.

For routing purposes, each node divides the tree into a collection of sub-trees. The first sub-tree is the highest sub-tree not containing that node. The second is the highest sub-tree in the remaining graph not containing the node, and so on. In Fig. 2, I, II and III represent the highest sub-trees corresponding to node D, in that order.

TABLE I

DISTANCE OF KEY 01101 FROM NODES

| Node | Distance |
|---|---|
| 10000 | 11101 |
| 00100 | 01001 |
| 01001 | 00100 |
| 01100 | 00001 |

Distances in the binary tree are measured by XOR'ing the two keys between which the distance is to be measured. By comparing the distance of a key from the identifiers of different nodes and selecting the node with the least distance, one can ensure that the data corresponding to the key is assigned to the node that lies in the same sub-tree as the key. As an example, Table I shows the distance of the key 01101 from the four nodes in the tree of Fig. 2. The node with identifier 01100 is the closest to the key and also lies in the same lowest sub-tree as the key. Thus, the definition of the term "closeness" in Kademlia is well established. The routing table of each node consists of at least one node from each of its corresponding sub-trees. This ensures that every node is reachable from a particular node within a reasonable distance of $O(\log N)$, the proof of which is beyond the scope of this paper.

## III. CompP2P: The Base Protocol

ComP2P is a distributed, peer-to-peer resource sharing platform in which participants can contribute to a pool of computational resources and, in turn, acquire access to the pool in order to perform tasks with intense computational requirements. Such a service requires an infrastructure that can ensure efficiency, reliability and an equal, fair load-sharing mechanism. Distributed hash tables provide an excellent infrastructure that satisfies these requirement. A brief discussion on how distributed computing and file sharing are not dissimilar from the point of view of DHTs follows.

DHTs are content agnostic - their worst-case performance or algorithms do not change with the input data. Hence, as long as the input data is a stream of varying binary streams, DHTs work correctly and efficiently. This property is very intuitively connected to file storage, but can be extended to distributed computing too. A computation task can be defined as a (*data*, *algorithm*) pair in which the data is a binary string and the algorithm is a sequence of operations to be carried out on the data. Computers understand algorithms as a sequence of encoded instructions in a specified format which are executed by a hardware processing unit. The encoding is invariably a binary format and hence the algorithm, too, can be expressed as a binary string. Thus, a computation task can be completely expressed as a binary string by concatenating the data and the algorithm strings[1].

[1] The data and the algorithm can be uniquely identified in multiple ways: by having fixed-sized fields for both or by having a marker pattern to separate the two. A marker pattern in the data or the algorithm can be prevented from being misinterpreted by using the technique of bit-stuffing.

```
<?xml version=“1.0” encoding=“UTF-8”?>
<comp2p>
   <header>
      <version>1.0</version>
      <type>TaskAssign</type>
      <origin>192.168.0.54:4004</origin>
      <seq><!-- 32 byte random number --></seq>
   </header>
   <algorithm>
      <format>java_bytecode</format>
      <encoding>base64</encoding>
      <type><!-- algorithm output type --></type>
      <value><!-- algorithm --></value>
   </algorithm>
   <data>
      <arg0>
         <type><!-- argument 0 type --></type>
         <value><!-- argument 0 value --></value>
      </arg0>
      <arg1>
         <type><!-- argument 1 type --></type>
         <value><!-- argument 1 value --></value>
      </arg1>
        .
        .
        .
   </data>
</comp2p>
```

Fig. 3. A typical ComP2P packet which consists of the protocol version number, origin and destination addresses and a 32-byte random sequence number in the header section. The algorithm section contains the format, encoding and the actual value. The data section is divided into numbered variables, each of which possesses a type and a value. The entire task is self-sufficient and does not depend on any other task for its result. A sequence number is required to identify packets at the origin node of the task when the result has arrived to correctly mark the corresponding outstanding request as completed.

A computation task must be completely defined by its data and algorithm. This enforces certain restrictions on the algorithm; namely, the algorithm should not depend on any other computation task[1] - the result of a computation should always be the same regardless of the state of the network or the configuration of the node that is performing the computation. This property of determinism is necessary if the output is to be correct, or indeed if the output is to be computable. The result is that the infrastructure needs to be designed keeping in mind that different architecture of different machines should not, in any way, affect the output of a computation task.

A ComP2P packet consists of the packet header and a single (or multiple, depending on the implementation) computation task enclosed inside an XML structure. A sample packet is shown in Fig. 3. A single computation task consists of an encoded algorithm and an input data vector. The output type is specified along with the algorithm, and may be a primitive or complex data type.

```
<?xml version=“1.0” encoding=“UTF-8”?>
<comp2p>
   <header>
      <version>1.0</version>
      <type>TaskReturn</type>
      <origin>192.168.0.201:4009</origin>
      <destination>192.168.0.54:4004</destination>
      <seq><!-- original sequence number --></seq>
   </header>
   <result>
      <format>java_bytecode</format>
      <encoding>base64</encoding>
      <hash><!-- MD5 hash of algorithm --></hash>
      <type><!-- algorithm output type --></type>
      <value><!-- algorithm output value --></value>
   </result>
</comp2p>
```

Fig. 4. A typical ComP2P task response that is returned by the assigned node asynchronously to the origin of the task. The packet consists of the output data type, the output value and a hash of the original algorithm for verification purposes.

An alternate implementation of ComP2P can be employed for scenarios where a large number of computation tasks share the same algorithm. In such a situation, the algorithm can be stored in a separate, parallel, file storage peer cloud consisting of the same nodes and based upon the same DHT structure. The key of the algorithm can then be passed in the computation task packet instead of the algorithm value. The caching property of Kademlia will ensure that the algorithm is stored on every node requesting for it, thus reducing the load on the node originally storing the algorithm [16].

The Kademlia protocol consists of four Remote Procedure Calls (RPCs): PING, STORE, FIND_VALUE and FIND_NODE. Of these, PING and FIND_NODE are required to maintain the structure of the overlay network, and need not be modified. In the file storage implementation, STORE commands the appropriate node to store the data passed as an argument to the RPC. In ComP2P, STORE commands the appropriate node to perform the computation required for the task passed to the RPC as an argument. After the computation is completed, the target node informs the origin node of the results of the computation asynchronously. A sample structure of such a packet is provided in Fig. 4. Due to the underlying DHT structure, this STORE command is guaranteed to find the appropriate target node that is assigned to perform the computation. The FIND_VALUE RPC is largely unused, but can be utilized to poll the target node for a status update in case of a time-out. Since the caching property of Kademlia causes FIND_VALUE to store the response in a local cache, leading to incorrect responses for future FIND_VALUE requests, this caching property needs to

[1] This limitation can be overcome by assigning a sequence of computation tasks to the same node and caching the results for a certain amount of time.

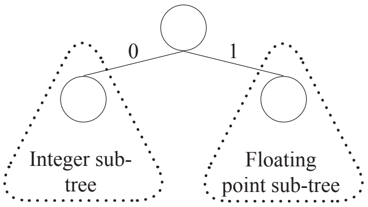

Fig. 5. The structure of a ComP2P binary tree with the integer nodes relegated to the integer sub-tree and the floating point nodes residing in the floating point sub-tree.

be disabled for correct functioning of this feature[1].

## IV. Extensions and Optimization

### *A. Exploiting Computation Aptitude of Nodes*

In distributed, peer-to-peer file storage infrastructures implemented using DHTs, all nodes are treated equal and are assumed to possess the same amount of resources; namely, bandwidth. Storage space is not an issue since a node is always considered to be capable of servicing a STORE command. If a node runs out of storage space, it can no longer service STORE commands and should ideally be treated as a failed node. A study of the average bandwidth available to a home user in the US reveals that 70% of home users have downstream speeds less than 6 Mbps and over 88% of households connected to the internet have upstream speeds less than 1.5 Mbps [13]. Consequently,  a majority of users are in a single speed bracket and hence the deviation in the capabilities of a node from the average is minimal and the assumption that all nodes can be treated equal is reasonable.

The computational power of a node depends primarily upon the architecture of the node and, to some extent, on the bandwidth available to the node. Let $C$ be the average computation time required per byte of data, $T$ be the number of bytes transmitted per second and $D$ be the total number of bytes in a particular computation task. Then, the fraction of time $\alpha$ spent on computation to the total time required to service a computation task, ignoring overhead losses, is:

$$\alpha = \frac{C.D}{C.D + \dfrac{D}{T}} \tag{1}$$

The size of a packet, $D$, can be treated as a constant. Also, the bandwidth available to a node does not vary significantly, hence, $T$ can be safely treated as a constant too. Thus, as the computation time per byte of data increases, the fraction of time spent on computation approaches 1. This shows that, in compute-intensive tasks, the computational power of a node is more important than its bandwidth. Also, as the computation time required per byte increases, the effect of bandwidth correspondingly decreases.

At an architectural level, instructions inside an algorithm can be classified into two categories: integer instructions and floating-point instructions [9]. These instructions invoke different functional units inside a processor. Consequently, the performance index of a node can be expressed as a pair of its integer performance and floating point performance. Various tools are available to benchmark different aspects of the computational power of a desktop processor [12]. The variation in these performance indices is large for different types of processors: general-purpose desktops tend to have balanced performance, whereas scientific desktops and servers focus on floating point performance [10]. Thus, the average time required to perform a computation task depends not only on the average performance rating of the node but also on the nature of the algorithm (integer or floating point) and the specialization of the node.

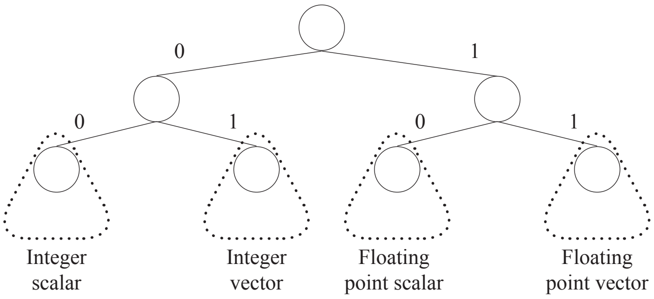

Fig. 6. A ComP2P binary tree with nodes classified as integer scalar, integer vector, floating point scalar and floating point vector nodes.

An analysis of the algorithm of a computation task will reveal the distribution of integer and floating point instructions. If the analysis shows that the algorithm depends heavily on either of the two categories, it is beneficial to perform the task on a node which has a better performance index for that category. This requires that the nodes in a peer cloud be classified broadly as "integer nodes" and "floating-point nodes" as per their relative strengths in the corresponding categories.

Consider the Kademlia binary tree of a typical ComP2P infrastructure in which the identifier of every node is prefixed with either 0 or 1 depending on whether the node is more capable at integer or floating point instructions respectively. The structure of such a tree is shown in Fig. 5.

Due to the unique binary tree-based structure of Kademlia, all nodes prefixed with the same prefix string reside in the same highest sub-tree corresponding to that prefix. Thus, the categorization of nodes into integer or floating-point nodes can be extended to multiple categories in a hierarchical order. For example, if, in both the categories shown in Fig. 4, nodes can also be categorized as either "scalar" nodes or "vector"[2] nodes, the result is a hierarchical binary tree as shown in Fig. 6.

As per the XOR metric of Kademlia, data is assigned to that node which is in the same lowest sub-tree as the data's key in the Kademlia tree [16]. Hence, by prefixing the key of a com-

[1] This caching property of the ComP2P DHT should not be confused with the caching property of the algorithm-storage DHT mentioned previously.

[2] "Vector" operations are also known as Single Instruction Multiple Data (SIMD) instructions. A popular example of an instruction set containing such instructions is Streaming SIMD Extensions (SSE) employed by Intel processors.

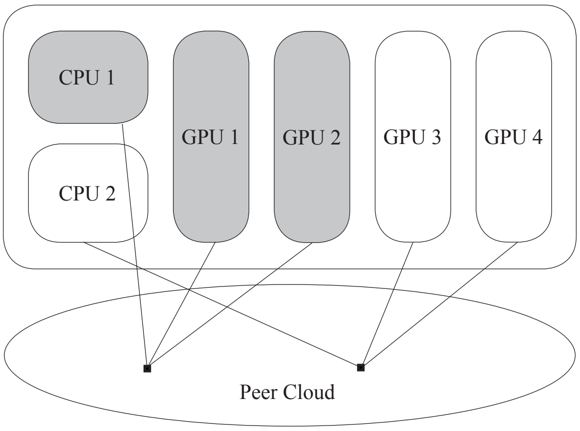

Fig. 7. A computer with two CPUs and four GPUs representing two nodes in a peer cloud. It is essential that each node have a minimum capability of performing any computation asked of it.

putation task with the same prefix as of the node, it is possible to control which sub-tree the task is assigned to. For example, as per Fig. 6, the key of a task requiring nodes from the floating point scalar sub-tree would be prefixed with "10".

It is important to note that, while this categorization skews the distribution of tasks which might violate fairness, the skew is restricted by the number of categories in the cloud. Once the destination sub-tree of a task is determined, the target node within that sub-tree depends on the hash of the task packet. Thus, load distribution in an even manner is maintained within a category sub-tree. This relaxation in the fairness criterion is reasonable as long as the number of nodes in all the sub-trees is fairly large and the tasks initiated inside the cloud are distributed evenly between all the categories.

### *B. Exploiting Disparate Performance of Nodes*

As mentioned earlier, the Kademlia DHT treats all nodes as equal with equal resources. While the discrepancy in performance in executing different types of instructions was addressed in the previous section, the variation in the overall computational power of nodes is yet unaddressed. This pronounced variation is caused by the availability of a large number of processors in the market in different performance segments as well as the disparity in performance between older generation processors and newer processors. This disparity is exacerbated by the exponential improvement in processor technology over the last three decades in accordance with Moore's law [11].

The deviation of a node from average performance increases when a node gains the ability to use an accelerator core. Accelerator cores are highly specialized processing cores separate from the central processing unit that are used to serve specific computation requirements. A common example is a graphics processing unit (GPU) which is used to render graphics for computer games. Traditionally, accelerator cores have had very limited instruction sets and hence proved unsuitable for general purpose computation. In recent times, however, with the advent of the NVIDIA CUDA technology, GPUs have been increasingly used in scientific computations and compute-intensive software. This is because GPUs are high-performance vector floating point computers whose computational power far outstrip the fastest desktop processors. As an example, the single precision floating point processing power of the fastest Intel processor (Westmere architecture) as of December 2009 was a little more than 130 GFLOP/s theoretically. In comparison, the single precision processing power of the NVIDIA Geforce GTX 480 (based on the GF100 architecture) launched around the same time was around 1375 GFLOP/s theoretically [18]. Ironically, the cost of a Geforce GTX 480 was around one-third the cost of the fastest Intel processor during that period. It is clear that, when leveraged properly, an accelerator core can greatly increase the computational power of a node. Also, unlike a processor, an accelerator core is not required continuously for the maintenance of the node and, therefore, can be utilized to a greater extent.

Additional power available at a node's disposal can be utilized by running multiple instances of the ComP2P application on the same computer. Thus, a computer is represented by multiple nodes in the peer cloud. For improved regulation of loading of a node, it is prudent to define a base node capacity, $\gamma$. $\gamma$ represents the amount of computational power an instance of ComP2P can consume from the local computational power available, and can be defined in terms of CPU cycles, instructions or energy. The value of $\gamma$ is a global constant. It should be small enough to accommodate the processing power of the slowest of all nodes, but not too small so as to crowd the DHT with too many nodes and degrade performance. Faster nodes wishing to donate extra resources to the pool may run multiple instances of ComP2P interacting with the cloud on different ports. This method also exploits the inherent multiprocessing capabilities of today's processors. However, each node on the peer cloud should be capable of performing any computation task asked of it since it is possible (though improbable in a large network) that sub-trees of other categories are empty in which case the responsibilty of performing the computation might fall on a node not optimized for that kind of computation. Fig. 7 shows a single computer with multiple processing units represented by multiple nodes in the cloud, with each node having at least minimum capability.

## V. Future Work

Though the base ComP2P protocol has been defined, a number of experiments and simulations are required to fine-tune the protocol for optimal performance. The size of a computation task has been left open to interpretation. For a smaller sized packet, the communication overhead becomes too large; conversely, a very large packet violates the principle of distributed computing. A balance needs to be struck based upon the needs and the visions of individual peer clouds.

While optimization regarding architectural differences in terms of integer and floating point instructions have been discussed, a different, more complex type of categorization based on predefined libraries of compiled code already installed in certain nodes is yet to be considered. A complex categorization would lead to a highly heterogeneous, diverse tree with clusters of nodes specialising in specific types of computations. Such a tree would require special heuristics to maintain fair load-sharing and reliability.

While accelerator cores greatly improve performance, they are still inherently limited in nature. An alternate accelerator core, called as Field Programmable Gate Array (FPGA), can be used to implement any algorithm as a digital circuit in real-time [21]. Such a system would show a tremendous improvement in algorithms written specifically for FPGA devices - the downside being the relatively high difficulty in implementing algorithms for FPGAs.

## VI. Conclusion

The world has seen a paradigm shift in the way a computer connected to a network is perceived. The network is no longer a mere interconnect between multiple, individual entities. Instead, a computer is now a gateway to a single, global, persistent entity, and the user interacts with this entity. Just as peer-to-peer file sharing is a worldwide phenomenon to share and disseminate with all mankind the knowledge possessed by our society as a whole, so also it is hoped by the author that peer-to-peer computing becomes a phenomenon where all the computers in the world pool their resources together to form an extremely powerful supercomputing entity to serve the needs of all mankind. ComP2P is but another step in the path to create the most powerful supercomputer ever conceived - the combined power of all computers in the world.

## Acknowledgment

The author gratefully acknowledges the assistance of Dr. Dheeraj Sanghi for his excellent teachings, helpful comments and suggestions.

## References

[1] B. Goldsmith, "CompTorrent: Applying BitTorrent Techniques to Distributed Computing". 2006. http://eprints.utas.edu.au/270/1/comptorrent_techreport.pdf

[2] B. Hari, K. Frans M., K. David, M. Robert, S. Ion, "Looking up Data in P2P Systems". 2003. http://www.cs.berkeley.edu/~istoica/papers/2003/cacm03.pdf

[3] B. Tony, "Google Outages Damage Cloud Credibility". September, 2009. http://www.pcworld.com/businesscenter/article/172614/google_outages_damage_cloud_credibility.html

[4] BitTorrent Concepts. Accessed 2011-04-23. http://www.bittorrent.com/help/faq/concepts

[5] BOINC Statistics. Accessed 2011-04-23. http://boincstats.com/stats/project_graph.php.

[6] BOINC, Open-source software for volunteer computing and grid computing. Accessed 2011-04-23. http://boinc.berkeley.edu

[7] E. Ramez, N. Shamkant, *Fundamentals of Database Systems*, 3$^{rd}$ ed., Addison–Wesley, ISBN 0-201-54263-3, 2000.

[8] F. Ina, "Sidekick Outage Casts Cloud over Microsoft". October, 2009. http://news.cnet.com/8301-13860_3-10372525-56.html

[9] H. John, P. David, *Computer Architecture: A Quantitative Approach*, 4th ed., Elsevier, ISBN 978-81-312-0726-0. p. 10.

[10] [9]. p. 13.

[11] [9]. p. 3. Fig 1.1.

[12] [9]. p. 30.

[13] Industry Analysis and Technology Division, Wireline Competition Bureau, "Internet Access Services: Status as of December 31, 2009". December 2010. http://www.fcc.gov/Daily_Releases/Daily_Business/2010/db1208/DOC-303405A1.pdf

[14] List of Distributed Computing Projects. Accessed 2011-04-23. http://en.wikipedia.org/wiki/List_of_distributed_computing_projects

[15] M. Nick, "Google Announces Chrome OS". July, 2009. http://www.pcworld.com/article/168028/google_announces_chrome_os.html

[16] M. Petar, M. David, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric". 2002. http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf

[17] NIST – Computer Security Division – Computer Security Resource Center. Accessed 2011-04-23. http://csrc.nist.gov/groups/SNS/cloud-computing

[18] NVIDIA CUDA C Programming Guide, Version 3.2. 2011-11-9. http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf

[19] S. Hendrik, M. Klaus, "Internet Study 2008/2009". 2009. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009

[20] S. Ion, M. Robert, K. David, K. Frans M., B. Hari, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In Proceedings of the ACM SIGCOMM '01 Conference. August 27-31, 2001. http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[21] W. Remigiusz, *Synthesis of Compositional Microprogram Control Units for Programmable Devices*. Zielona Góra: University of Zielona Góra. ISBN 978-83-7481-293-1. pp. 153.