Node Capability Aware Replica Management for Peer-to-Peer Grids

A Vijay Srinivas and D Janakiram
Distributed & Object Systems Lab,
Dept. of Computer Science & Engg.,
Indian Institute of Technology, Madras, India.
http://dos.iitm.ac.in
Email: avs,djram@cse.iitm.ernet.in

Abstract

Data objects have to be replicated in large scale distributed systems for reasons of fault-tolerance, availability and performance. Further, computations may have to be scheduled on these objects, when these objects are part of a grid computation. Though replication mechanism for unstructured P2P systems can place replicas on capable nodes, they may not be able to provide deterministic guarantees on searching. Replication mechanisms in structured P2P systems provide deterministic guarantees on searching, but do not address node capability in replica placement.

We propose Virat, a node capability aware P2P middleware for managing replicas in large scale distributed systems. Virat uses a unique two layered architecture that builds a structured overlay over an unstructured P2P layer, combining the advantages of both structured and unstructured P2P systems. Detailed performance comparison is made with a replica mechanism realized over OpenDHT, a state of the art structured P2P system. We show that the 99th percentile response time for Virat does not exceed 600 ms, whereas for OpenDHT, it goes beyond 2000 ms in our test-bed, created specifically for the above comparison.

I. INTRODUCTION

Replication is an important aspect of scale in distributed systems. Data objects have to be replicated in large scale distributed systems for reasons of fault-tolerance, availability and performance. Further, distributed computations may have to be scheduled on these objects. These may arise from compute

This work was done as part of the PhD work of Vijay at IIT Madras, under Prof. Janakiram. Vijay is currently with Oracle and can be reached at: vijay.agneeswaran@oracle.com.
intensive applications or for data processing. In this context, it may be useful to store the replicas on capable and lightly loaded nodes. Thus, a scalable node capability aware replication scheme would be very useful for large scale distributed systems and is the focus of this paper. This paper becomes more relevant with the advent of the Data Intensive Super-computing (DISC) systems (see http://www.cs.cmu.edu/~bryant/pubdir/cmu-cs-07-128.pdf).

Computational science applications typically operate on data that may be distributed across organizations, for instance see [1]. A typical application is in astronomy, where a world-wide virtual telescope is being built [2]. The idea is to form a collaboration of several observatories in different countries. Provenance data, which means data derived from the raw data need to be archived and stored. The aim of the International Virtual Observatory Alliance (www.ivoa.org) is to provide access to such data to both professional and non-professional astronomers to enable new discoveries. Thus, hundreds to thousands of geographically distributed users need access to the data for performing computations.

Another interesting application that involves data sharing in the large comes from the tele-medicine area. In order to provide timely advice of expert doctors to patients in remote villages, the patient data such as an ECG must be sent to the nearest available health specialist. The patient’s historical information may be required by the doctor to make his judgement on the patient’s condition and offer advice. The advice can be shipped back to the patient in the remote village centre. Similar data sharing requirements are also emerging in other areas such as genomics (see [3]) and physics (see [4]). Data (provenance data, in the case of astronomy applications or game state in the case of MMOG or patient data) may have to be replicated to improve availability. So, there is a need to replicate the data at appropriate locations to handle node/network failures, minimise computation time and/or bandwidth as well as to provide wide accessibility.

Peer-to-Peer (P2P) systems have addressed issues of scalability and fault tolerance. But they cannot be used directly, as argued below. Unstructured P2P systems such as Gnutella [5] or Freenet [6] have addressed fault-tolerance issues and have also scaled up quite well. They use a random graph as the overlay and use flooding or random walks to search for data. The advantage of unstructured P2P systems is that they can support complex queries (for instance, select nodes with storage capacity greater than 5GB). However, they do not provide deterministic guarantees about the search (even whether data will be found).

Structured P2P systems such as Tapestry [7] or Pastry [8] use Distributed Hash Tables (DHTs) or other data structures as the overlay. The overlay is formed based on node identifiers. Objects are also given identifiers from the same space. DHTs map the object identifier with a node identifier that is
responsible for information on that object. Hence, they provide $O(\log(n))$ guarantees for object searches. However, they are limited to exact queries. The neighbourhood of a node is determined based on node ids in structured P2P systems, whereas unstructured P2P systems allow any application specific criteria for neighbourhood formation. In large data grids, nodes must be able to form neighbours based on capabilities, which may be difficult with structured P2P systems.

A key issue in designing a replica management system for grids is to provide the ability to create replicas on nodes with given capabilities, so that computations scheduled on the replicas are efficient. The state of the art DHT deployment named as OpenDHT [9] exemplifies this issue. The performance of OpenDHT was slow (99th percentile response time was order of a few seconds) on Planetlab due to a few nodes being slow (stragglers). There were efforts to improve the performance by using delay aware and iterative routing [10]. However, it is difficult to “cherry pick a set of well performing nodes for OpenDHT” as quoted in that paper. We attempt to solve this problem, by forming the structured layer over an unstructured layer that provides nodes with required runtime capabilities.

A few efforts have made been to introduce node capability aware clustering of the peers in P2P systems. These include the topology and heterogeneous routing scheme described in [11] which builds a topology and node capability aware overlay and SemPeer [12], which provides a modelling mechanism for semantically clustered peers. However, these efforts only target unstructured P2P systems, implying that they cannot provide non-probabilistic guarantees on the search. To the best knowledge of the authors, this work provides the first instance of integrating structured and unstructured P2P systems to build a node capability aware replica management system.

We propose an efficient solution to the replica management problem in grids by a unique two layered architecture. The unstructured layer forms neighbours based on node capabilities (in terms of processing power, memory available, storage capacity and load conditions). We have developed an advertising protocol that allows the changes in load conditions to be propagated up to k-hop nodes. Object replicas are stored in the unstructured layer based on node capabilities. A structured DHT is built over this unstructured layer by using the concept of virtual servers. That is, a node which has been found to be capable is made to join the DHT with a virtual node identifier with a closest matching id as the node it replaces. This enables searches based on object identifiers to be returned in $O(\log(N))$ time.

A. Contributions

This paper makes the following contributions:
• This paper proposes the idea of integrating structured and unstructured P2P systems by using virtual servers to achieve node capability aware efficient replica management. The idea of building an application specific structured layer over an underlying unstructured layer could serve as a building block for distributed systems of the future, where possibly millions or billions of data objects may have to be handled along with distributed computations on them.

• Extensive performance studies have been done to demonstrate the above ideas. The paper also compares Virat with OpenDHT to illustrate the following: Stragglers or undesirable nodes could cause performance degradation in OpenDHT, whereas Virat eliminates stragglers and achieves good performance.

B. Scope of the Paper

The platform presented in this work works well for small objects. It may not be particularly suitable for large objects (order of Gigabytes in size), which are addressed by data grid efforts such as [13]. Special mechanisms for update operations on replicas such as function shipping or incremental updates may have to be explored. Security issues are also outside the scope of this paper. A trust model could be incorporated into Virat in order to realize a trusted environment that we assume. This is however, left for future research. We also assume that there is no free riding (Free riding is a problem that originated in P2P file systems where peers only access files but not store or give storage space to the system [14]). Free riding can be prevented by incorporating economic models such as [15] or incentive schemes such as [16] into Virat.

C. Paper Organization

The rest of the paper is organised as follows: Section II explains the two layered architecture from a functional perspective. Section III explains the replica management framework that we propose. Section IV details the shared object space realization over the proposed two layered architecture. Section V provides implementation details, while section VI substantiates our ideas with detailed performance analysis, including performance comparison between Virat and OpenDHT. Section VII compares the proposed node capability aware replication middleware with the state of the art in the literature. Section VIII concludes the paper and provides directions for future research.

II. A TWO LAYERED ARCHITECTURE FOR P2P GRIDS

We have motivated the need for a two layered architecture in order to build a replica management solution for P2P grids. This section describes the functionality of the two layers. Figure 1 pictorially
depicts the two layered architecture.

The unstructured layer provides proximity and capability based clustering of resources. In other words, the unstructured layer groups the nodes of the system into proximity based zones or clusters. Within a zone, nodes maintain neighbour lists. The nodes add other nodes with higher Horse Power Factor (HPF) [17] as neighbours. The HPF is a measure of capability of a node, in terms of computation, memory and communication and more importantly, storage capacity. The neighbour list of a node is altered dynamically, since HPF changes dynamically. The unstructured layer implements a HPF propagation algorithm, which propagates dynamic HPF values across nodes. It also implements a heart beat algorithm to detect node failures. Both the above algorithms may result in the neighbour lists being updated across the nodes. Details of the HPF propagation algorithm and the heart beat algorithm can be found in [18].

A few mechanisms have been proposed to aggregate HPF information in the unstructured layer, such as Astrolabe [19]. Astrolabe provides a scalable information aggregation mechanism using publish-subscribe architecture. However, at the time of producing this paper, the Astrolabe software was not available to the authors. Hence, we used a simpler heuristic to advertise HPF information to other nodes:

1) Advertise $HPF$ value within a logical hop of radius $k$; where $k$ is either a prespecified system parameter or a parameter that is calculated based on initial configuration of the overlay topology.

2) For each node $j$ that is its logical neighbour, node $i$ checks if $HPF_i$ is greater than $HPF_j$ and if so, then the $HPF_i$ value is propagated to node $j$. Node $j$ might propagate it further to its neighbours

Fig. 1. Two Layered P2P Architecture: An Illustration
with \( HPF \) values lesser than \( HPF_i \).

3) The above process is carried out up to a maximum of \( k \) logical hops.

The structured layer enables the platform to handle node/network failures. The information required to recover from failures is stored in this layer. The information gets a quasi-random identifier and is replicated in \( k \) other nodes of the system. The structured P2P layer ensures that if even one of the \( k \) nodes is alive, the information can be recovered in \( O(\log(n)) \). The \( k \) replicas are maintained in the same cluster/zone. This makes it easier to maintain consistency among the \( k \)-replicas\(^1\). The details of the routing and how it enables recovering from failures can be found in an earlier developed system Vishwa [18]. Vishwa uses two layers for a computational grid and uses the structured layer for storing information for recovering from transient faults. Virat constructs the structured layer out of the capable nodes from the unstructured layer for storing the replicas. Virat uses virtual servers for gluing the structured layer with the unstructured layer by moving nodes up and down between the two layers.

### III. Dynamic Replica Management in Virat

In this section, we explain the key idea, namely the use of virtual nodes to integrate structured and unstructured P2P systems. This makes Virat a generic platform for providing data related grid services over the Internet. It also enables capability based replica placement on nodes.

A Virat runtime object is present in every node of the system. This object serves as the interface for the client code to access the Virat services. When the Virat runtime object gets a request for creating a replica of an application object, it interacts with the Object Meta-data Repository (OMR) service and returns a replica of the object to the client. The OMR maintains information about objects including current replica locations of each object. The OMR also caches a copy of each object in order to make replicate requests from clients faster.

A. Using Virtual Servers to Integrate Structured and Unstructured P2P Systems for Replica Management

We achieve node capability aware replication by using the concept of virtual servers to integrate unstructured and structured P2P systems. A node in the structured layer may become loaded\(^2\). This node leaves the structured layer. Another capable node, chosen from the unstructured layer is given the same

\(^1\)This is one reason why proximity based zones are maintained. The other reason is that computing elements can be clustered closer, resulting in faster communication between the computing elements.

\(^2\)It must be noted that load on a machine is measured using the standard metrics such as CPU queue length [20] and consequently have lower Horse Power Factor (HPF) values. Moreover, load could be caused by other jobs running on the node. This is likely, as we are dealing with a non-dedicated set of machines forming a P2P grid.
virtual identifier as the original node and is made to \textit{join} the structured layer. All DHTs support the \textit{join} and \textit{leave} operation and so, can realize virtual servers. The concept of virtual servers was first introduced in Chord [21] and has been used for load balancing. This paper shows it is feasible to use virtual servers for replica management.

We employ mechanisms outside the DHTs to move application objects (OMR data) to ensure that the application still functions correctly.

The overlay construction (structured layer over an unstructured layer) will be explained below. When a node contacts the bootstrapping component (zonal server) to join the overlay, it is returned with a proximal neighbour list. Based on capabilities and network delays, the node chooses its neighbours from this list, forming the unstructured layer. The DHT gets formed on top of this as the node sends a join message with its randomly generated hashed (using sha1) identifier (id). This message gets routed to the node with the closest matching id. This node as well as nodes along the path send the routing tables and prefix matching entries respectively. The joinee node thus builds its routing tables and sends it to the nodes in its routing table. A Chord-like routing protocol to ensure that zones are not isolated is also used. Further details of the routing can be found in the implementation section.

It is important to note that the criteria for node movement from unstructured layer to the structured layer is a HPF threshold. However, a single threshold for all applications and for all environments may not be the right approach. Instead, we let the application developer to fix the threshold for the particular application and a specific run. Moreover, the HPF calculation can be varied, by giving more weightage to a particular component - a developer interested in a global storage system may give more weightage to this component, rather than the computing power. Consequently, the number of nodes which move up to the structured layer can be tuned by the application developer. If the threshold is too low, then too few nodes may move up. By increasing the threshold slightly, more nodes can be made to form the structured layer. Care must be taken so that too many nodes do not move up due to high threshold values, as not all of them may be best performing.

It can be argued that the overhead of moving large number of small objects or a small number of large objects (that is required for the \textit{join} and \textit{leave} operations, may become significant. This opens up the issue that our approach seems to be a coarse grained load balancing compared to the fine grained approach of [22], [23]. We counter this argument by pointing out that coarse grained applications (such as scientific applications on the grid) can make use of more capable and less loaded nodes at runtime, in spite of the associated overheads. These applications typically may run for days together, whereas the time required for a node to move to the structured layer from the unstructured layer may be only order of
a few minutes. Thus, our approach is more suited to coarse grained long running applications common in grids.

B. More Replication Details

1) Replica Consistency: A replica, in order to perform a write operation on a given object, first performs a local write operation. It then retrieves meta-data for this object’s id from the structured layer, through the local routing component. It thus gets location of other replicas of this object. It picks a random subset of the replicas and with a probability based on a probability function, propagates the update to the subset. This is done recursively, with each of the other replicas getting the meta-data, picking the subset and with a probability sending updates to the subset. This constitutes the push phase of the algorithm. If a replica was disconnected and reconnects, it starts a pull phase by contacting on-line replicas and getting updated based on version vectors. This is an adaptation of a probabilistic update protocol based on epidemic rumour spreading [24]. This would work well for applications which can tolerate probabilistic guarantees and is pragmatic in a P2P setting. It must be noted that we do not make any contribution with respect to consistency in a wide-area setting, but only use an existing algorithm.

2) Replica Location: OMRs maintain meta-data on objects, including data on current replica locations. A request for replica location (originating from a read or replicate request) arrives at the OMR of the originating node. It must be observed that every node acts as an OMR and the OMRs form a structured overlay. Thus, the originating OMR forwards the request to the appropriate destination OMR. The originating OMR achieves this by using the route method of the underlying DHT and passing the oid from the replica location request. The DHT maps the oid to a node responsible for storing meta-data relating to that oid. Thus, the destination OMR returns the meta-data to the originating OMR, which in turn returns the replica location to the application. The P2P organisation of the OMR is thus different from the hierarchical organisation of Replica Location Indices (RLIs) in Giggle [25]. This is similar to the P2P Replica Location Service proposed in [26]. However, they do not address the formation of zones as is done in Virat. The zones themselves form a Chord ring, to ensure routing can be done across zones.

3) Meta-data VS Data Replication: The OMR maintains a copy of the object, in addition to storing meta-data about it. Thus, it stores both data and meta-data. The OMR data is automatically replicated in k-nodes of the structured layer - these will be typically capable lightly loaded nodes (only such nodes move from unstructured layer to structured layer). As long as all the k-nodes do not fail simultaneously, a copy of the replica and its meta-data is guaranteed to be reachable. For each oid, the meta-data includes current replica locations - which could be OMRs that cache or store copies of this object. Meta-data
could also be information about other nodes which create replicas explicitly. The meta-data replication is important, especially in grids. If meta-data is not replicated and is unreachable, clients would not be able to connect to a replica, even though it is alive.

4) Inter-zonal Replication: The k-replicas of meta-data and data for a given object are stored in the leaf set nodes of the structured layer. This means that the replicas are all within the same zone. This makes consistency protocols efficient. Also, it would be useful in a grid computing context to have replicas on proximal nodes (which characterises the zone). However, the replication is also done across zones to handle the case when the entire zone becomes disconnected. This is achieved by replicating OMR data in some of the non-leaf set nodes. It must be remembered that non-leaf set neighbours are formed from other zones (refer joining protocol of the implementation section). This is done to ensure zones are not isolated from each other.

Thus, most replicas are within the same zone, while there may be some replicas across zones. It must be noted that application is free to create replicas on any node (intra or inter-zone) as may be required. These nodes will be added to the meta-data for that object. This may be required for realizing say a Massively Multi-player Online Gaming (MMOG) (such as the one in http://butterfly.net) application over Virat.

5) Ratio of Nodes in the Two Layers: An important factor in the two layered architecture is the ratio of nodes in the structured layer and the unstructured layer. A HPF threshold determines if a node can move up to the structured layer. If the threshold is set too low (high HPF), most of the nodes will move up, resulting in straggler presence in the structured layer. If the threshold is set too high, there is the danger of having too few nodes in the structured layer. Setting appropriate threshold is important and is done best by the application developer, who can vary the threshold in each run and get appropriate number of nodes in the structured layer after a few trials.

It can be argued that since the system uses a HPF threshold for movement of nodes up and down the layers, it could thrash if the load fluctuations are frequent. However, thrashing can be avoided by setting two thresholds: an upper threshold - which decides node movement up to the structured layer and a lower threshold - which decides node movement down to the unstructured layer. A node at the upper layer would not move down till it reaches the lower threshold, while a node in the unstructured layer would not move up till it crosses the upper threshold.
IV. VIRAT: A TWO LAYERED P2P ARCHITECTURE BASED RECONFIGURABLE AND SCALABLE SHARED OBJECT SPACE

In this section, we show how a shared object space abstraction is realized over the two layered P2P architecture explained in the previous section. We start with a basic Intranet version of Virat. Then we explain its scaling as a loosely structured P2P system by integration with Pastry. Finally, we present the fully P2P version of Virat. We effectively present three versions of Virat as they evolved. The idea is to capture the design decisions that motivated the use of a two layered P2P architecture compared to a naive distributed version or the use of a loosely structured P2P architecture.

A. Intranet Scale Shared Object System

A DSM runtime object is present in every node of the system. This object serves as the interface for the client code to access the DSM services. When the DSM runtime object gets a request for creating shared objects, it interacts with the DSM services, namely the lookup and Object Metadata Repository (OMR) services and returns a replica of the object to the client. The OMR maintains information about objects and the current accessors for each object. It is responsible for propagating updates to the replicas. Each object has a data counter, for ensuring data centric CC [27].

The DSM runtime on each node handles the initial object discovery requests. It looks up the object in its cache, failing which it contacts the lookup service of the cluster. If the object has not been created before or has been created in a different cluster, the DSM runtime sends a request to the OMR. The OMR creates a new unique oid for the object and gives back a copy of the object. The OMR maintains list of current accessors for each object. When an update request message is received from an accessor, it is propagated to all other accessors, the order depending on the application specific consistency criteria.

B. Virat Over Pastry

Nodes as well as networks are more prone to failure in an Internet setting, making the initial version of Virat unsuitable. This is true especially with respect to OMR failures. OMRs maintain information about objects or object meta-data, including details like object id and list of accessors for the object. Each cluster has a pre-designated node as an OMR. To handle OMR failure, a lookup server was used. The lookup server keeps information about the current location of its (cluster level) OMR. The assumption was that both the lookup server and the corresponding OMR do not fail simultaneously. This may be untenable in an Internet setting. If an object that is created in one cluster (its meta-data is maintained by OMR1) needs to be replicated in a different cluster (which has OMR2), OMR1 does not know which

October 29, 2008 DRAFT
of the OMRs has meta-data corresponding to that object. Hence, OMR1 searches through all the OMRs and finds that OMR2 maintains this data. This causes replica creation requests to be delayed, especially in a wide-area setting. Thus, a better mechanism is required to handle OMR failure and efficient object lookups across clusters.

The OMRs form a Pastry ring and route data through the routing protocol of Pastry in order to find which OMR maintains information about a given object. This enables the lookup time to be only $O(\log(n))$ for $n$ nodes and can reduce cross-cluster replication request times considerably. Information maintained in any OMR is also stored in k-replicas, meaning k other OMRs. Thus, even if an OMR fails, the routing protocol of Pastry ensures that, among the k OMRs that are alive at that point, the message is routed to the OMR with the closest matching id. Thus, OMR failures are also handled elegantly.

It must be noted that this architecture (OMRs forming P2P overlay and routing on behalf of other clients) is similar to loosely structured P2P systems. An example is the reference [28], which gives the design challenges and guidelines for super-peer based data management. Consequently, this design of Virat has the advantages of those systems, including efficient searches compared to centralized or naive distributed approaches. However, it also suffers from the problems associated with super-peer systems - difficulty in handling super-peer failures through redundancy and dynamic layer management. In other words, if an OMR fails, its clients are disconnected and would not be able to lookup objects, unless they are configured with the address of backup OMRs. Further, since OMRs form a structured overlay, the meta-data gets replicated in random nodes, not necessarily proximal. This leads to inefficient meta-data consistency maintenance. The problem arises due to the fact that structured P2P systems do not address locality in neighbour selection [29], though they address locality in routing. Complete details of the super-peer Virat, built over Pastry can be had from [30]. The problems of selecting neighbours based on locality and creating replicas on capable nodes is handled in this paper by the two layered P2P architecture.

C. Fully P2P Virat: A Shared Object Space Abstraction

All nodes form the P2P overlay, instead of only OMRs forming an overlay as in the previous design [30]. Moreover, all nodes have routing capabilities and are peers in this sense. As explained before, the unstructured layer forms proximal neighbours based on node capabilities, including storage and computing power. It also has a mechanism for advertising capability information, implying that the neighbourhood set on each node varies dynamically, as load and storage capacity of nodes vary. Any node in the zone can play the role of an OMR. There are also k-replicas for each OMR in the structured layer. All k
replicas are in the same zone (implying they are proximal - with respect to standard measures such as hop count). The following paragraphs explain the working of the shared object space over the two layered P2P architecture.

The client program instantiates a Virat Client Interface (VCI) object and makes invocations on it to create, replicate or write shared objects that could be located anywhere. The client program invokes `makeObjectSharable` method on the Virat client interface object to `out` (in Linda parlance [31]) an object into the shared object space. When the VCI object receives this method call, it creates an identifier (id) for this object. This is a unique id formed by hashing (by sha-1) the zone id concatenated with node IP address and a random number. It sends a `route` message with this id. The `route` message ensures that the message is routed to the node with the closest matching identifier that is alive. This message reaches a node playing the role of the OMR in that zone. This OMR stores meta-data about this object including the location of the replicas. It also ensures that the meta-data is replicated in k other OMRs of the zone. Thus, the meta-data can be retrieved if all k-nodes do not fail simultaneously (a standard assumption in P2P systems). Further, the structured layer ensures that this meta-data can be retrieved from anywhere (even from other zones) given only the object id.

The client program invokes a `replicate` method on the VCI object to create a replica of an object, given its id. It must be noted that this method can be invoked on any node of the Internet. On getting the `replicate` request, the VCI object informs to the local routing component (route manager) to retrieve meta-data for this object id. The route manager ensures that the message is routed to one of the k-OMRs that are currently alive and has the closest matching id as this object id. This is achieved by the structured layer. The OMR with the closest matching id stores meta-data about this object and gets a copy of the object. It passes the copy to the originating VCI, which passes the replica to the application.

When a `replicate` request is made from a node outside the zone, the VCI object first routes the request to the node with closest matching id within the originating zone. This node routes the request to the corresponding node in the destination zone. This can be either a direct routing if the destination zone has registered with the originating zone or an indirect routing. It must be noted that each node registers with some nodes at distance $2^k$ in the node id space. The idea behind nodes having neighbours from $2^k$ distant zones, is that a structured routing algorithm similar to Chord [21] can be realized across zones. The complete algorithm for routing across zones can be found in [18]. Thus, a `replicate` request can be made from anywhere and Virat would ensure that the object is replicated at the destination node.

The case of the VCI object receiving a `write` request to write into an object with a given id is explained below. The VCI object retrieves meta-data for this id from the structured layer, through the local routing
component. It thus gets location of other replicas of this object. It picks a random subset of the replicas and with a probability based on a probability function, the update is propagated to the subset. The details of the update algorithm were explained in the replica consistency section earlier.

V. IMPLEMENTATION OF THE TWO LAYERED ARCHITECTURE

The structured network used for intra-zonal routing is based on Pastry, while that used for inter-zonal routing is based on Chord. For Intra-zonal routing, an important concern is efficiency - Pastry can ensure routing is done to the node responsible for a given identifier (ID) and is nearest (in terms of network distance). For Inter-zonal routing, the key concern is to ensure zones do not become disconnected. We exploit the key property of the Chord routing protocol that even if very few routing table entries are correct (in the face of churn), correct (but possibly slower) routing can be guaranteed - implying slower routing would still work across zones, preventing zone disconnection even in the face of churn.

The virtual servers and object metadata repositories are run in each node (individual peers) of the system. However, only one node must serve as the zonal server for a zone. Moreover, this node (zonal server) is used only for bootstrapping and does not play any role in routing. The overall architecture of Virat and a simple routing example is explained by the figure 2.

A. Bootstrapping and Routing in the Two Layered Architecture

The details of the bootstrapping and routing in the two layered architecture can be found in [18]. The two layered architecture of Vishwa and Virat have commonalities and consequently, Virat uses Vishwa in its implementation. The main difference between Virat and Vishwa is that Virat uses virtual nodes to move nodes from one layer to another, while in Vishwa there are no virtual nodes. The way the two layers are constructed and routing within the layers is common. We summarize these details below. A node must contact the closest available zonal server for joining the system. Zonal servers generate node identifiers and enable nodes to know each other and form clusters. When contacted by a node for joining, the zonal server returns a list of neighbours based on node capabilities. The joinee node chooses its neighbours from this list. The zonal server also returns a set of proximal zonal servers to the joinee. This is to ensure that zones are not isolated from each other - the joinee gets neighbours by contacting other zonal servers returned by the bootstrapping zonal server. The Distributed Hash Table (DHT) is also constructed at the initialisation phase. The algorithm for routing table initialisation can be summarised as below and is depicted in the figure 3.
The joinee node contacts closest node with a join message. This message is routed to the node with the closest matching id. The routing table is constructed from the routing table of this node (with closest matching id) and routing entries from nodes along the route. In order to route across zones, the joinee node also sends remote join messages to other zonal servers and updates its routing table accordingly. The joining protocol has been captured in the Figure 3. A routing algorithm similar to Chord [21] has been realized across the zones. The complete details of the routing can be found in Figure 4.

B. Replication Service (RS)

A replication service that caters to data grids has been designed over the two layered architecture. The RS provides *replicate* and *remove*, two main services to its clients. Typically, end user applications or
grid schedulers can programmatically use the RS to replicate data for reasons of performance (to reduce computation time or minimise bandwidth) or fault tolerance. Meta-data is also maintained by the RS and its consistency is ensured. The RS also provides a lookup service that returns the current locations of a replica. Another important service provided by the RS is named as `getPossibleReplicaLocations`. The RS uses node capability and proximity as a heuristic to return a list of possible locations for replicating a given object. It must be noted that RS uses the two layered virtual server based architecture to realize all its services. The two layered architecture maintains capability information in the form of a neighbour list of nodes that are also proximal in the unstructured layer. This information is utilised by the RS to return a set of possible locations for replicating an object. The RS also uses the `replicate` method to have an object replicated on a specified node. Consistency of the replicas as well as meta-data of the replica
VI. Performance Studies

This section details some of the performance studies we have conducted over Virat to show its scalability and usability in a grid environment. The initial set of experiments have been conducted on an Institute wide network, consisting of about fifty five heterogeneous machines, each having memory from 64MB to 1GB, processing speed from 350MHz to 2.4 GHz and running different operating systems (linux, solaris etc.). The machines are spread across three clusters, with each cluster being connected by a 10/100 Mbps Ethernet connection. A few nodes from the University of Melbourne, Australia were used and a wide-area test-bed was formed. This test-bed was used for the wide-area results reported in this paper.

Fig. 4. Two Layered Architecture: Routing Across Zones
A. Virat and OpenDHT: A Replica Management Comparison

OpenDHT is a state-of-the-art DHT deployment from Berkeley/Intel research. OpenDHT provides the storage abstraction. This means it could be treated as a distributed shared storage space. However, the other differences should not be neglected. In this section, we have compared and analysed the response times for both OpenDHT and Virat on our Institute as well as the Internet test-beds.

First, we have redeployed OpenDHT on our test-beds. Even though a deployment is available on PlanetLab, this must be done in order to make a comparison with Virat on similar test-beds (Since we did not have access to PlanetLab, we were not able to deploy Virat on PlanetLab).

We use the response times given in terms of percentiles as in [10]. This gives a better picture than giving only the mean, as the mean could be affected by extreme values. Figure 5 gives the comparison of response times on the Intranet and Internet test-beds for read requests (replica creation in a new node). As can be observed from the table, till the 50th percentile, the response times for OpenDHT are lower or comparable to Virat. However, the 90th and 99th percentile response times for OpenDHT are very high, compared to Virat. Even on the Intranet, the 99th percentile for OpenDHT goes up to 124 ms, compared to just 53 ms for Virat. This gets magnified on the Internet, as the 99th percentile response time for OpenDHT is over 2000 ms, compared to just 600 ms for Virat. The Internet result for OpenDHT is comparable to those given in [10].

![Graph showing response time comparison between OpenDHT and Virat on Intranet and Internet test-beds.](image)

Fig. 5. Response time Comparison with 1000 Objects: Intranet and Internet Results

The main reason for poor worst case performance of OpenDHT is the presence of stragglers (slow nodes or heavily loaded nodes). Nodes with low capabilities in terms of processing power and/or RAM

---

5Mainly, a byte storage space is a lower level abstraction to the programmer compared to a shared object space.
could be stragglers. Nodes with high capabilities could become stragglers when their HPF values go above 40. This has also been captured in [32]. The node, with 256MB RAM (the one whose HPF went up to 50), has acted as the straggler in this experiment\(^4\). Contrastingly, the unstructured layer of Virat chooses the neighbours based on HPF. Consequently, the node with 256MB RAM never moves to the structured layer. Any other node in the structured layer whose HPF exceeded 50 also drops from the structured layer. Thus, Virat avoids the problem of stragglers and is hence able to achieve significantly better 90th and 99th percentile response times compared to OpenDHT.

Another reason for the 99th percentile performance improvement of Virat over OpenDHT is the way get and put requests are realized in OpenDHT. A put request is forwarded to eight leaf-set neighbours of the node with the closest matching id as the object id. These nodes occasionally synchronise with each other. However, a get request is sent to all the neighbours. Responses from six nodes are combined (if they have not synchronised with each other) before a get request is answered to the client. This also causes the worst case performance to drastically degrade. Contrastingly, in Virat, the read request is routed to the currently alive node with the closest matching id. The synchronisation happens during a write request. This is based on the invalidation protocol or probabilistic update protocol explained before. A read request returns as soon as a response is received from one replica. This makes worst case performance of read requests to be better in Virat compared to OpenDHT. However, write requests may be more expensive.

1) Overhead Comparison on a Larger Test-bed: We have repeated the above studies on a larger test-bed, which comprise a hundred and fifty virtual nodes connected through a Gigabit backbone at EPFL. The virtual nodes are formed from 100 physical nodes by paravirtualization. The physical nodes all have quad processors with 8GB RAM. The results are given in figure 6. It can be observed that the results are similar to the earlier results on the Intranet test-bed at IITM, with OpenDHT values being slightly worse. This is because of the use of virtual nodes, which implies higher straggler load.

2) Join Overheads Comparison: Virat and OpenDHT: A comparison of the join overheads of Virat and OpenDHT is given in figure 7. For this studies also, the 150 node test-bed at EPFL was used. It can be observed that the join overheads for Virat are substantially higher compared to OpenDHT, with Virat taking nearly 650 milliseconds for node joining on a 100 node network, while OpenDHT takes only 30 milliseconds. The main reason for this is that Virat is designed to work in a low churn environment efficiently, while OpenDHT is designed to operate in a high churn environment. Virat spends time in

\(^4\)By instrumenting the DHT routing code, we could trace the DHT path for a given object identifier. The straggler took as much as 90\(\text{ms}\) in the worst case to complete its processing, compared to only about 10 \(-\) 15\(\text{ms}\) for other nodes in the path.
forming the network, making sure neighbours (especially inter-zone neighbours) are physically close - this is done by using ping messages. Moreover, this overhead also includes the HPF propagation overheads. While in OpenDHT, which is designed to operate in a high churn environment, there is little time spent in identifying proximal neighbours. Since the churn is high, it would make no sense in spending lot of time in network formation, as nodes could leave and that information would have to be updated again.

The other question that may arise is the use of ping for network formation. The use of network coordinates such as Vivaldi [33] have been tried in previous studies such as [34], which show that they could have a negative impact on node lookup performance under churn. However, the use of ping was simpler to implement and is the main reason for its use in Virat. The use of network coordinates such as Vivaldi or GNP [35] and the exploration of other techniques to reduce the join overhead in Virat is left for future research.

B. Performance of Virat under Load Fluctuations

An important question that arises is that of the performance of Virat under load fluctuations. Since, Virat uses HPF threshold to move nodes up and down, we have done experiments to test its performance under load fluctuations. This set of experiments was also conducted on the EPFL test-bed mentioned
earlier. The CPU queue length was used as a measure of load. The results are given in figure 8. The first row shows the performance of Virat that was observed initially. It can be inferred from the first row graphs that since Virat uses only a single HPF threshold to move nodes up and down the two layers, load fluctuations result in unnecessary node movement. That is, five of the nodes which had a fluctuation, were moved down to the unstructured layer and then moved back again to the structured layer. This led us to investigate the possibility of using two thresholds in order to avoid such behaviour\(^5\). We have henceforth modified the code to use two thresholds, TU for the upper threshold, which is used for node movement down to the unstructured layer and TL for the lower threshold for the node movement to the structured layer (it must be remembered that the a low HPF value suggests more capable node, not less).

The bottom row shows the result under similar conditions. In this case, due to the use of the two thresholds, Virat does not move the nodes unnecessarily and only moves down the nodes whose HPF goes above the TU, upper threshold - indicative of heavy load. By fixing the TU and TL appropriately, one can avoid load spikes from causing unnecessary node movement between the layers, preventing thrashing-like behaviour.

\(^5\)We are grateful to Prof. Ramakrishna, of Aerospace Engineering at IITM, who gave us this suggestion.
C. Overhead of Node Movement from Unstructured to Structured Layer

The overhead of moving a node from the unstructured to the structured layer must be measured. The move operation involves a

- leave operation: when the old node leaves the structured layer due to high load or other reasons;
- meta-data movement: moving the meta-data maintained by the old node to the new node;
- join operation: when the new node joins the structured layer with the id of the old node.

The leave operation is an asynchronous operation, in that the routing state gets updated asynchronously. Thus, it takes constant time, (of the order of only a few milliseconds) equal to a route method overhead.

In the results that follow, the overhead of node movement from unstructured to structured layer is hence calculated as the sum of join operation and overhead of meta-data movement. The overhead of moving the OMR meta-data from one node to the other has been measured for varying object sizes and with varying number of objects hosted by each OMR. This is captured in the form of a three dimensional graph in figure 9.

---

**Figure 8:** Virat Performance under Load Fluctuations: The use of Two Thresholds

- Single Threshold $T = 30$
- Upper Threshold $T_U = 30$
- Lower Threshold $T_L = 15$
- Horse Power Factor = HPF
- No. of Nodes in Structured Layer = $N_S$
- No. of Nodes in Unstructured Layer = $N_U$
It can be observed that if the OMR maintains object copies in addition to the object meta-data, the overhead of data movement could be significant, especially at higher scales. Hence, the Virat middleware has been optimized so that the OMR maintains object copies and meta-data for small number of objects and discards the object copies at higher scales. Consequently the system scales better at higher scale reflected by the graceful performance degradation. If the OMR maintains only meta-data, it returns the meta-data on a search request. The peer OMR gets the replica location from the meta-data and obtains an object copy. This results in increased overhead at higher scale, degrading the performance.

The join overheads have been measured for increasing number of nodes in the system and is captured in figure 10. The two sets of readings are for the proximal neighbour selection within the same zone using ping messages. The second simply uses the triangulation law, without ping messages, similar to Pastry. If a node is at a distance of say \( x \) from the zonal server (bootstrapping component), the zonal server applies the triangulation inequality and returns nodes that are within \( x \) distance from itself. As can be observed from the figure, the overhead of using ping messages becomes too high and has been dispensed with henceforth. The reason being that though it results in more accurate neighbour selection, it may not scale. At higher scales, it may be better to sacrifice accuracy for performance.

It can be observed that the join overhead is roughly around 1 second. Thus, the overall overhead of a node moving from unstructured to structured layer is about 2 seconds. This overhead is not an insignificant overhead. However, it has been observed in a few grid deployments such as TeraGrid (http://www.teragrid.org) that load fluctuations are not very drastic. This implies that nodes may not
be moving from one layer to the other too frequently. Under this assumption and for coarse grained tasks (where grain size is big enough so that computation dominates communication overheads), the proposed approach works well.

The number of message exchanges that are required for a join operation was also measured. This result is summarized in figure 11. It can be observed that even with 10000 nodes, it takes only maximum of nine messages for a join operation.

D. Significance of Replica Placement for Grid Task Performance

We illustrate impact of replica placement on grid task performance in figure 12. Neutron Shielding [36] is a nuclear physics application, used to determine the number of neutrons penetrating through a given thickness of lead sheet. The experiment predicts this number using Monte-Carlo simulation. In our experiments we assume a lead sheet of 5 units thickness and vary the number of neutrons in powers of 10. This is a compute intensive application. The lower and upper bound values for each of the tasks are stored in the replicas. The figure shows that significant reduction in computing time and hence speedup are achievable as a result of the efficient replica placement. This experiment was conducted by running Vishwa software on each of a cluster comprising twenty nodes. The computing tasks are managed by Vishwa, including task splitting and runtime task management. Virat is used for the data management part only. The size of the replicas are small, of the order of kilobytes only. The idea of exploiting Vishwa for managing computation and Virat for managing replicas can be exploited to execute complex computations.
that involve replicas. Further exploration along this thread is left for future work.

It must be noted that these performance studies have been done under controlled conditions with low churn rates. It has been argued elsewhere [37] that Bamboo DHT on which OpenDHT is based, has been designed to handle high churn rates. If churn rates are low, it may not result in optimal performance. However, it is intended to achieve robustness under high churn rates, corroborating our studies. The observation that Bamboo DHT is designed for robustness but not optimality has also been made in [38].

E. Simulation Results: Varying Straggler Load and Percentage of Requests Through Straggler with Large Number of Nodes

One of the key difficulties in conducting controlled large scale experiments in distributed systems is that the nodes are non-dedicated. This implies that it is difficult to control the load conditions on the nodes (several users may spawn tasks on the nodes). Secondly due to the physical limitations, it may be hard to get access to and conduct experiments on thousands of nodes. Hence, the scale of experiments can be increased by using a simulation. It is important to ensure that the simulation approximates the real system to the closest extent possible. We have used data from real experiments on our Intranet and Internet test-beds and used these as inputs to the simulator.

Churn refers to routing table and other changes necessary to handle node dynamics in structured P2P systems. By low churn rates, we mean that node sessions typically last for at least tens of minutes and nodes do not leave within few seconds.
A home-grown simulation test-bed was created using Simjava [39] as the basis. Simjava is a process based discrete event simulator written in Java and available as an open source software. The simulator has been built similar to the way Gridsim (http://www.buyya.com/gridsim) was built over Simjava. Each node was modeled as a Simjava entity, which would be realized as a separate thread. Nodes can be connected by named ports. To simulate thousand nodes, an equal number of threads were created. The inter-node delays between the entities were given as application level delays, measured from the actual Intranet/Internet test-bed. This is different from giving the delays at the networking level and simulating network level routing as in NS2 or other network level simulators. The key simulation parameters have been captured in figure 13. The DHT routing was emulated and based on the object identifier specified in the lookup request, the entities were connected along the path. Entities were also given identifiers, similar to the way node identifiers are given in a structured P2P system.

The first set of experiments was conducted by simulating thousand nodes. The response time for object lookup, similar to the read requests in the earlier experiments, was measured. It is assumed that the DHT routing takes four hops in this case. The path could go through a straggler or it could go through only normal nodes. The nodes along the path could be in the same LAN (if both source and sink are in the same LAN and DHT routing does not go across LANs) or the nodes could be in adjacent LANs (in case of an Intranet test-bed). The various cases are captured and the response time results for the Intranet test-bed simulation are given in figure 14. The figures give the response time results for varying straggler...
node loads, with the straggler node delay coming from real measurements. The load was measured using the *top* call in Linux, which measures the CPU queue length. The response times are displayed stem-like, with the mean and 90th and 99th percentile response time being the three circles in each stem. It can be observed that as straggler loads increase, the worst case response time also increases and could go up to nearly 1200 milliseconds in an Intranet environment. This is the advantage of the simulation, as the load on the straggler could not be controlled and varied so much in the real measurements given in the previous section.

Another simulation parameter, the ratio of requests not passing through the straggler, was kept fixed at 0.8 in the above experiments. This implies that only about 20% of requests go through the straggler. This parameter can be varied, to check the response time when more (or less) requests pass through the straggler. The results of these set of experiments are captured in figure 15. It can be inferred that as more requests pass through the straggler, the mean and worst case response times increase. This is because of the longer delays introduced by the straggler and its bottleneck effects. The worst case response time is not so much impacted compared to the mean. This is because even if more requests pass through the straggler, the processing delay for individual events is nearly the same and is independent of the number of events through it (We did not include simultaneous events in our experiments). Thus, it can be said...
that we have measured the best case performance of the straggler, with its worse case performance being when there are concurrent requests. This analysis is confirmed by observing the results plotted in figure 16. The figure shows the response time for OpenDHT when there are concurrent requests. Concurrent requests are generated by reducing the interval between events from $500 - 700 \text{ms}$ to only $10 - 20 \text{ms}$. Contrastingly, concurrent requests do not degrade the performance of Virat drastically. This is captured in figure 17. It can be observed that with interval of $10 - 20 \text{ms}$ between requests, the response time is an order of magnitude lower in Virat compared to OpenDHT. Even with more concurrent requests (reducing the interval between events), the response time only degrades gracefully. This can be attributed to requests queueing up, as service time (processing time at each node) is comparatively higher than the rate at which requests are generated.

The effect of straggler was also studied in an Internet environment by assuming one of the links (source node to sink or any intermediate links) to be a Wide Area Network (WAN) link. In this case, the WAN delay was calculated from the Internet test-bed WAN link between IIT Madras and University of Melbourne. Thus, a random distribution in the range of $300 - 400 \text{ ms}$ was used as the WAN link delay in the simulation. The response time was measured and captured in figure 18.
Fig. 15. OpenDHT Intranet Response Time: Varying Ratio of Requests Through the Straggler

Fig. 16. OpenDHT Response Time: Concurrent Requests
Contrastingly, Virat avoids stragglers by using the two layered architecture and moving nodes up and down. This results in Virat response time being nearly a constant and is captured in table I.

We have also simulated a network consisting of ten thousand nodes. In this case, the DHT routing takes on average 5 hops. It must be noted that with a thousand nodes, the routing was taking only 4 hops, reflecting its logarithmic property. The key difference in the two cases is the possible presence of more than one straggler. If there are two stragglers and a request happens to pass through both, it may be delayed. The results for varying loads on the two stragglers with 64% of the requests passing through both stragglers with all the nodes (source, sink and nodes in the DHT routing path) in the same LAN is captured in figure 19. The case when the different nodes are in adjacent LANs within an Intranet is captured in figure 20. The case when there is one WAN link (Internet) is captured in figure 21. These results show that if the load on both stragglers is high (5 - CPU queue length as per uptime/top), any request that passes through both could be delayed by as much as 2500 ms. This is similar to the results
TABLE I

<table>
<thead>
<tr>
<th></th>
<th>Intranet Case (msecs)</th>
<th>Internet Case (msecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>48</td>
<td>394</td>
</tr>
<tr>
<td>90th Percentile</td>
<td>57</td>
<td>428</td>
</tr>
<tr>
<td>99th Percentile</td>
<td>60</td>
<td>450</td>
</tr>
</tbody>
</table>

obtained when running OpenDHT on Planetlab, as reported in [10]. The paper [10] attempts to optimize the OpenDHT routing by routing around stragglers. The worst case performance occurs when the straggler becomes involved in routing, similar to the results we obtain.

![Graphs showing response time with varying load](image)

Fig. 19. OpenDHT Object Lookup Overhead with 10000 Nodes and 2 Stragglers with Source, Sink and Intermediate Nodes in same LAN

VII. RELATED WORK

Virat is contrasted with other P2P replica management systems including P2P file sharing systems and storage systems. We then compare our approach with some P2P techniques that have been proposed for load balancing in DHTs. We proceed to compare Virat with approaches for data management in grids.
Replication issues have been investigated in the context of unstructured P2P systems in [40] and in [41]. The paper [40] proposes replication strategies for unstructured P2P systems. Uniform replication keeps a constant replication factor for all objects, irrespective of its popularity. Proportional varies the replication factor based on the popularity - popular data are replicated more. They also propose an optimal strategy to minimize the expected search size for successful queries. By improving query response time, the paper [41] suggests that dependability of a P2P system can be improved indirectly. The replication strategy includes factors such as available space and load conditions. It can be observed that both these efforts only are targeted at unstructured P2P systems. Since Virat builds a structured overlay over an unstructured layer, popular data can be replicated on capable nodes.

Issues in dynamic replication over WAN environments including grids and P2P systems have been brought out by [42]. The key issues include ownership, active and passive replication, keeping track of replicas, determining the number of replicas for a hot data item, popular and un-popular data items and sharing complex data types. While Virat does not address the first two, it addresses most of the other issues. The replicas of a given object are maintained in OMR as meta-data. Though this data is not
Fig. 21. OpenDHT Object Lookup Overhead with 10000 Nodes and 2 Stragglers with Source, Sink and Intermediate Nodes in Internet with only one WAN Link

guaranteed to be up to date, it is good hint. Similarly, using meta-data, the number of replicas for a hot data item could be determined. Popular data can be replicated on capable nodes. Complex data types can be encapsulated by using objects and can be shared. We have experimented with data up to a few hundred Megabytes and have found that Virat performs reasonably (refer performance studies section of this paper).

Structella [43] is the only other effort that we are aware of that tries to integrate structured and unstructured P2P system concepts. Structella builds Gnutella over a structured overlay. It retains the content placement and file discovery mechanisms of Gnutella, but takes advantage of structure to optimise queries. Specifically, it uses the broadcast mechanism available in MS Pastry to send queries to overlay nodes. However, structella does not place replicas on capable nodes, an important issue in grid computing systems.

P2P storage management systems such as PAST [44], Oceanstore [45] and Ivy [46] attempt to provide sophisticated storage management systems over P2P overlays. Ivy has been proposed as a read/write P2P file sharing system. PAST provides a persistent caching and storage management layer on top of Pastry.
All the three storage systems (Ivy, PAST and Oceanstore) have been built over DHTs. In the words of [29] virtualization (through DHTs) "destroys locality and application specific information". Though it may be possible to build application specific data placement criteria into a DHT, it is a non-trivial task. We provide one way by using an unstructured P2P layer. It may not be easy to achieve replica placement considering node capabilities using only a DHT. Moreover, We have also shown in the performance studies section that even a state of the art DHT can be ineffective due to the presence of stragglers, while the goal of Virat is to improve performance by eliminating the stragglers.

OpenDHT is a open source DHT deployment that is shared across various applications by possibly untrusted clients. It supports a simple get/put interface for getting and putting byte storage for a given identifier. It is realized over Bamboo DHT(bamboo-dht.org), that is similar to Pastry but has differences in handling node dynamics. First, OpenDHT is not a shared object space. The level of abstraction provided to programmer is different. For instance, programmer has to take care of object serialisation, RTTI (runtime type inferencing) etc. to realize an object storage on top of the byte storage that OpenDHT provides. The performance of OpenDHT (especially worst case response time) suffers due to the presence of stragglers or slow nodes. This has been improved by using delay aware and iterative routing in [10]. In contrast, the unstructured layer in Virat chooses a set of nodes based on node capabilities and load conditions. The structured layer is formed by this set of nodes by using virtual nodes. Further, load dynamics are handled by allowing virtual nodes leave/join the structured layer. Thus, at anytime, the structured layer consists of nodes with the best capabilities and load conditions. This makes Virat an ideal platform for building data grids, whereas it may be difficult to use OpenDHT in a similar context.

JuxMem attempts to unify P2P systems and Distributed Shared Memory (DSM) [47]. It uses a cluster manager (any node can play this role) to manage memory made available by nodes within that cluster, with the cluster managers forming a P2P overlay. JuxMem uses advertisements to propagate information about memory resource providers. It is realized over Juxtapose (JXTA) (http://www.jxta.org), an open source P2P platform. The main disadvantage of JuxMem is that it uses the JXTA multicast primitive as the basis for ensuring consistency. This may be unreliable. It also needs total ordering, due to which it may not scale up. Recent efforts have been made to implement atomic multicast using consensus protocols [48]. However, consensus protocols may also be expensive, especially in a P2P setting. Moreover, JuxMem has been evaluated only in a cluster environment. It may be difficult to realize data related grid services over JuxMem. This is because JuxMem does not provide the object space abstraction, but only provides storage space abstraction. Further, Virat can create replicas on capable nodes so that when a grid task is scheduled, it would be efficient. This may not be possible with JuxMem currently. Juxmem which is
built over JXTA, does not have the two-layer abstraction and consequently the best nodes may not be chosen for replication.

A recent effort named PeerCast [49] builds a churn resilient end systems multicast protocol. PeerCast uses landmark based network proximity information to cluster nodes in order to minimize multicast delay. It also uses a virtual node based mechanism based on node capabilities to balance the multicast load. However, we see a limitation of PeerCast as the following: It uses all nodes for DHT routing (all nodes are peers), which implies that it could also be prone to the straggler effect, which we have demonstrated in OpenDHT. In contrast, Virat eliminates stragglers from the Overlay, by its two layered architecture. Moreover, PeerCast has only been evaluated using a simulation, while we have fully implemented the Virat prototype and performed experiments on several real test-beds.

A. Load Balancing in DHTs

The use of virtual servers was first proposed in Chord [21] to make the number of object ids supported by each node to be closer to the average. Virtual nodes have also been used for load balancing in P2P systems in [22], [23]. The paper [22] achieves load balancing by moving virtual servers from heavily loaded to lightly loaded nodes using join/leave operations of the DHT. The paper [23] has similar goals, but adds proximity awareness to guide load balancing decisions. In both these efforts, virtual servers are used to evenly distribute the DHT identifier space over the nodes. Thus, a single physical node may host more than one virtual server. Load balancing is achieved by moving virtual servers across nodes of the DHT.

In all of the above efforts, virtual servers are used to evenly distribute the DHT identifier space over the nodes. Thus, a single physical node may host more than one virtual server. Load balancing is achieved by moving virtual servers across nodes of the DHT. While the use of virtual servers is for load balancing within a DHT in Chord, the virtual servers concept is used to maintain the most capable nodes in the DHT in Virat. In other words, the whole structured layer is virtualized in terms of best performing nodes in Virat, while the virtual servers in Chord only partition the DHT identifier space for load balancing.

Load balancing in DHTs has also been achieved by using hints in the adaptive replication protocol of [50]. The adaptive replication protocol allows replicas to be placed on lightly loaded nodes that are not necessarily in the path of the query source to destination. The normal routing is augmented with hints to find the new replicas. Hints could also be incorporated into Virat as a search optimisation technique.
B. Data Management in Grids

Globus [51] a de-facto standard toolkit for grid computing systems, relies on explicit data transfers between clients and computing servers. It uses the GridFTP protocol [52] that provides authentication based efficient data transfer mechanism for large grids. Globus also allows data catalogues, but leaves catalogue consistency to the application. The paper [53] explores the interfaces required for a Replica Management Service (RMS) that acts as a common entry point for replica catalogue service, meta-data access as well as wide area copy. The RMS is centralised and may not scale up.

The reference [54] provides a P2P resource discovery scheme for grids. It extends the flock of condors [55] static resource discovery to handle dynamic discovery by using structured P2P system concepts. The central managers of individual flocks form a P2P overlay (similar to Pastry) while nodes within a flock form another P2P overlay to handle central manager failures. Our work can be considered as the data grid equivalent of the above paper, since we handle data and meta-data replicas and their consistencies also, in addition to resource discovery.

Replication strategies for data grids have been studies in [56]. The different strategies include best client (client makes many requests, beyond a threshold), cascading replication (store at clients along the path to the best client), caching (any requesting client gets a copy) and fast spread (store along path to every client). It assumes replicas are read only and does not consider update costs. Further, it only evaluates (based on simulation) the various replication strategies, whereas Virat provides a comprehensive replica management framework. Moreover, the paper [56] does not consider node capabilities in placing replicas - which is a key idea in Virat. Finally, the architecture is assumed to be client-server, in contrast to the P2P approach of Virat.

A P2P approach for building the Replica Location Service (RLS) has been proposed in [26]. It makes the observation that the grid services are currently being provided from the *interiors* and must be provided from the *edges* to tolerate failures and scale up in an Internet environment. However, it does not address optimal (or near optimal) replica placement in grids. Moreover, the RLS realization is based on Kademlia [57], a structured P2P system. This makes it difficult to achieve replica placement on nodes with desired capabilities, as argued before.

A high performance storage service for grids has been proposed in Distributed Active Resource arChitecture (DARC) [58]. DARC uses a hierarchical architecture with a top level "root" directory. The architecture is essentially client-server, with "root" directory information being maintained in *reliable* servers. The assumption of interior reliable servers is again made. This is different from the P2P approach
of Virat, where services are provided from the edges, in a possibly failure prone environment.

Thus, we are making a unique attempt at providing a scalable replica management middleware that can create replicas on capable nodes by using the concept of virtual servers to integrate structured and unstructured P2P systems.

VIII. CONCLUSIONS

We have presented a fault-tolerant and scalable replica management framework for P2P grids that allows replicas to be placed on nodes with desired capabilities, enabling computations scheduled on the replicas to run efficiently. This was mainly achieved by the two layered P2P architecture that glues the structured P2P layer with an underlying unstructured layer so that the best performing nodes move up to the structured layer. We have also performance bench-marked Virat against OpenDHT over both Intranet and Internet test-beds (created specifically for the comparison). This clearly demonstrates that the 99th percentile response time overheads of Virat is only 600ms, compared to over 2000ms for OpenDHT on the Internet test bed.

Virat essentially virtualizes the structured layer on top of an unstructured layer which may allow nodes to join only the unstructured layer and the best performing nodes to move up to the structured layer. This, we believe, could become an important way of constructing large scale distributed systems in general.

The use of virtual servers to integrate structured and unstructured P2P systems opens up research beyond the key issue that we have explored, replication. For instance, it is well known that structured P2P systems have difficulty in handling complex queries. The use of virtual nodes to integrate structured and unstructured P2P systems can make complex queries easier. The query could be executed in the unstructured layer and the node with the desired result could be moved up to the structured layer (through the virtual node). However, the challenge is to provide deterministic guarantees on the search. One possibility is to use the cached results in the structured layer to amortise cost of further queries.

The two layered architecture provides a unique way to integrate P2P systems and grids. The two layered architecture can also be viewed as providing a better (than existing realizations of grids) realization for grid services using P2P concepts. This is in contrast to other approaches which attempt to realize P2P applications over grid services, such as [59]. Future research directions include exploring fundamental theoretical foundations for P2P grids. We are also exploring query processing and schema issues to make Virat a P2P object data base management system. This includes supporting for storing and retrieving meta-data in Resource Description Format (RDF). This would be a step towards making Virat a semantic P2P system.
ACKNOWLEDGEMENTS

We acknowledge Dr. Rajkumar Buyya, University of Melbourne and Dr. Maluk Mohamed, MAM College of Engineering Trichy, for providing us with nodes for the Internet test bed. We also wish to thank the Department of Science & Technology, Government of India for sponsoring part of the research project. We also thank Reddy for help in the implementation and the other members of the DOS Lab for the comments on the work and on the paper. The authors also wish to acknowledge the help of Sravani Kurakula, the intern from BITS in Siemens, who helped in the simulation. Last, but not the least, the authors wish to thank the anonymous reviewers for their comments, as this helped improve the overall quality of the paper.

REFERENCES


[27] D Janaki Ram, N S K Chandra Sekhar, and M Uma Mahesh, “A Data-Centric Concurrency Control Mechanism for


Vijay Srinivas Agneeswaran obtained the Ph.D degree from the Indian Institute of Technology, Madras in 2008 under the guidance of Prof. Janakiram. He also completed his post-doctoral research from the Swiss Federal Institute of Technology, Lausanne (EPFL) and has since joined Oracle as Principal Researcher/Member, Technical staff. His research interests span grid and cloud computing as well as software engineering and data management. He has published in leading journals and conferences in these areas and has filed several patents with the US and European patent offices. He is a member of the IEEE and the ACM. He has also been selected in the list of Marquis list in the World 2009 edition.

D Janakiram is currently a professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Madras, India. He obtained his Ph.D degree from IIT, Delhi. He heads and coordinates the research activities of the Distributed and Object Systems Lab at IIT Madras. He has published over 30 international journal papers and 60 international conference papers and edited 5 books. His latest book on Grid Computing has been brought out by Tata Mcgraw Hill Publishers in 2005. He served as program chair for 8th International Conference on Management of Data (COMAD). He is the founder of the Forum for Promotion of Object Technology, which conducts the National Conference on Object Oriented Technology(NCOOT) and Software Design and Architecture (SoDA) workshop annually. He is the principal investigator for a number of projects which include the grid computing project from Department of Science and Technology, Linux redesign project from Department of Information Technology, Middleware Design for Wireless Sensor Networks from Honeywell Research Labs and A Mobile Data Grid Framework for Telemedicine from Intel Corporation, USA. He has taught courses on distributed systems, software engineering, object-oriented software development, operating systems, and programming languages at graduate and undergraduate levels at IIT, Madras. He is a consulting engineer in the area of software architecture and design for various organizations. His research interests include distributed and grid computing, object technology, software engineering, distributed mobile systems and wireless sensor networks, and distributed and object databases. He is a member of the IEEE, the IEEE Computer Society, the ACM, and a life member of the Computer Society of India. He can be reached at djram@iitm.ac.in. See also www.cs.iitm.ernet.in/ djram