Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 115–130. http://www.scpe.org
ISSN 1895-1767

DATA MANAGEMENT IN DISTRIBUTED SYSTEMS: A SCALABILITY TAXONOMY

A VIJAY SRINIVAS AND D JANAKIRAM*

Abstract.
Data management is a key aspect of any distributed system. This paper surveys data management techniques in various distributed systems, starting from Distributed Shared Memory (DSM) systems to Peer-to-Peer (P2P) systems. The central focus is on scalability, an important non-functional property of distributed systems. A scalability taxonomy of data management techniques is presented. Detailed discussion of the evolution of data management techniques in the different categories as well as the state of the art is provided. As a result, several open issues are inferred including use of P2P techniques in data grids and distributed mobile systems and the use of optimal data placement heuristics from Content Distribution Networks (CDNs) for P2P grids.

1. Introduction. Data management is an important facet of distributed systems. Data management encompasses the ability to describe data, handle multiple copies (replication or caching) of data objects or files, support for meta-data as well as data querying and accessing. Different approaches for data management have given importance to these different aspects and provide explicit support, while other aspects are implicitly or indirectly supported. For instance, Distributed Shared Memory (DSM) systems and shared object spaces handled consistency of replicated data, but supported meta-data indirectly through object lookups.

Orthogonal to the above mentioned issues of managing data, the main non-functional challenges are fault-tolerance, scalability and security, as illustrated in [32]. We survey various distributed systems from the perspective of scalability of data management solutions and provide a scalability taxonomy. We classify data management approaches into three categories: Centralized/Naively Distributed (CND) techniques, Sophisticated/Intermediate Data (SID) management techniques and Large Scale Data (LSD) management techniques. We give a brief view of the evolution of data management in each of the categories.

CND techniques for data management were used by DSM systems such as TreadMarks [10], Munin [25] and shared object spaces such as Linda [24], Orca [36] and T Spaces [4]. Many of these systems provide application transparent replica consistency management. They use centralized or naively distributed components to achieve the same. For instance, T Spaces uses a centralized server for consistency maintenance and for object lookups, while Java Spaces [81] uses a centralized transaction coordinator.

SID techniques have been used mainly in data management in grid computing systems such as [51], which provides a Replica Management Service (RMS). Some of these systems are characterized by data sharing across autonomous organizations at intermediate scale (possibly thousands of nodes). These approaches mainly manage replicated data in a grid computing environment. Data grids [27] handle data management as first class entities in addition to computation issues. They are characterized by the size of the data sets, which could be order of gigabytes or even terabytes. High Energy Physics (HEP) applications such as GriPhyN [31] and CERN [79] are examples of data grids. Other approaches that use SID techniques include Content Distribution Networks (CDNs) and data management in distributed mobile systems. CDNs such as Akamai [43] have been proposed to deliver web content to users from closer to the edge of the Internet, enabling web servers to scale up. Data management in distributed mobile systems are characterized by data sharing in the presence of mobile nodes, exemplified by systems such as Coda [74]. The common feature across these different systems is the scale of operation (thousands of nodes) that distinguishes SID techniques for data management. Many of these systems assume that failures are rare and reliable servers (distributed, not centralized) are available.

LSD management techniques do not assume reliable servers. The distinguishing feature of LSD techniques is that the execution of services is delegated to the edges of the Internet, resulting in high scalability and fault-tolerance. LSD techniques work well over the Internet and could handle millions of nodes/data entities. Peer-to-Peer file sharing systems such as Napster [57] and Gnutella [33], P2P file storage management systems such as PAST [15] and Oceanstore [49] as well as P2P extensions to Distributed DataBase Management Systems (DDBMS) such as PIER [38] and PeerDB [60] all fall into the LSD category.

A taxonomy of data grids has been provided in [87]. It compares data grids with related data management approaches such as CDNs, DDBMS and P2P systems. A functional perspective of data management that focuses on data location, integration, sharing and query processing as well as the different P2P systems that

*Distributed & Object Systems Lab, Dept. of Computer Science & Engg., Indian Institute of Technology, Madras, India, http://dos.iitm.ac.in,{avs, djram@cs.iitm.ernet.in}

address these functionalities is given in [50]. A survey of P2P content distribution has been provided in [77]. It examines P2P architectures from the perspective of non-functional properties such as performance, security, fairness, fault-tolerance and scalability. Our survey is broader and tries to provide the equivalent survey for grids, P2P systems, CDNs and DDBMS. We also provide a scalability taxonomy that distinguishes our survey from others. Further, we discuss state of the art in several of these areas and discuss how ideas/concepts/techniques from one area can be applied to others. The reader must keep in mind that though the authors have made an effort to be unbiased, the survey has limitations as it is perceived through their looking glass.

The rest of the paper is organized as follows. Section 2 discusses the CND techniques for data management and includes DSMs and shared object spaces. Section 3 discusses the SID techniques and includes data management in grids, CDNs, and distributed mobile systems. Section 4 discusses P2P data management techniques. Section 5 explores the state of the art data management techniques in distributed systems. Section 6 concludes the paper and includes a taxonomy figure and gives directions for future research.

2. CND Techniques: Data Replication in DSMs and Shared Object Spaces. DSM provides an illusion of globally shared memory, in which processors can share data, without the application developer needing to specify explicitly where data is stored and how it should be accessed. DSM abstraction is particularly useful for parallel computing applications, as demonstrated by TreadMarks [10]. Collaborative applications such as on-line chatting and collaborative browsing would be easier to develop over a DSM.

Page based DSMs can be more efficient, due to the availability of hardware support for detecting memory accesses. But due to the larger granularity of sharing, page based DSMs may suffer from false sharing. Relaxed consistency models including Release Consistency (RC) and its variants such as lazy RC allow false sharing to be hidden more efficiently than strict consistency models [64]. Munin [25] was an early DSM system which focused on reducing the communication required for consistency maintenance. It provides software implementation of RC. TreadMarks [10] is another DSM system that provides an implementation of release consistency. Java/DSM [91] provides a Java Virtual Machine (JVM) abstraction over TreadMarks. It is an example of page based DSMs, similar to Munin and TreadMarks.

Release consistency is a widely known relaxed consistency model for DSMs. Memory accesses are divided into synchronization (sync) and non-synchronization (nsync) operations. The nsync operations are either data operations or special operations not used for synchronization. The sync operations are further divided into acquire and release operations. An acquire is like a read operation to gain access to a shared location. A release is the complementary operation performed to allow access to the shared location. Acquire and release operations can be thought of as conventional operations on locks. There are two variations of RC, RC_{sc}—which realizes sequential consistency and RC_{pc}—which realizes processor consistency. RC_{sc} maintains program order from an acquire to any operation that follows it, from an operation to a release and between special operations. RC_{pc} is similar, except that write to read program order is not maintained for special operations. Eager RC, as the original RC became subsequently known [48], requires ordinary shared memory access to be performed only when a subsequent release operation is due by the same processor. Lazy RC (LRC) is a variation of RC in which processors further delay performing modifications until subsequent acquires by other processors and modifications are made only by the *acquiring* processor. LRC intuitively assumes competing shared accesses to be separated by synchronization operations.

2.1. Shared Object Spaces. Object based DSMs (also known as shared object spaces) alleviate the false sharing problem by letting applications specify granularity of sharing. Examples of object based DSMs include Linda [24], Orca [36], T Spaces [4], JavaSpaces [81] as well as an object based DSM in the .NET environment [75]. Orca relies on an update mechanism based on totally ordered group communication to serialize access to replicas. Even though a study has shown that the overhead of totally ordered group communication affects application performance minimally [37][1], the study was done on a Myrinet cluster. Orca has not been evaluated on the Internet scale. T spaces is a shared object space from IBM [4] that adds database functionality to Linda tuplespace [24] and is implemented in Java to take advantage of its wider usability. In addition to the traditional Linda primitives of *in, out, read*, T spaces supports set oriented operators and a novel rendezvous operator called *rhonda*. Global shared objects [90] allows heap objects in a JVM to be shared across nodes. Based on memory access patterns of applications, it also proposes various consistency mechanisms to be realized efficiently. However, it uses locks and per-object lock managers for keeping replicas consistent. It does not address failures of the lock manager. Java Spaces specification from Sun [81] provides a distributed persistent

[1] This is due to its choice of which objects to replicate—those with high read/write ratios and efficient implementation of totally ordered group communication.

shared object space using Java RMI and Java serialization. It provides Linda-like operations on the tuple space and uses Jini's transaction specification to achieve serializability of write operations. It also does not address fault tolerance, an important issue for Internet scale systems.

2.1.1. Globe. Globe [3] attempted to address the challenges of building software infrastructure for developing applications over the Internet. A key design objective of Globe was to provide a uniform model for distributed computing. This means that Globe provides a uniform way to access common services (such as naming, replication and communication) without sacrificing distribution transparency. Objects in Globe encapsulate policies for replication, migration, etc. Each object comprises multiple sub-objects, allowing an object to be physically distributed. The different sub-objects of an object include one each for semantics (functionality), communication (sending/receiving messages), replication and control flow. This helps the programmer to separate functionality from orthogonal non-functional properties such as replication. Objects also help in realizing distribution transparency by hiding implementation details behind well defined interfaces. The implementation framework of Globe is flexible, meaning that different implementations of the same interfaces are possible. It also provides an efficient mechanism for object lookups by using a tree based hierarchical naming space. It must be observed that distributed object middleware such as CORBA [61] also provide similar services such as naming and trading. But they cannot provide object-specific policies that can be provided in Globe.

2.2. Software Availability and Usage Summary. To the knowledge of the authors, T spaces and Java Spaces are widely used and are available as open source software. Linda is a specification and has been implemented by several groups. Orca and Globe are research prototypes, information on their deployment and use is not available.

2.3. Observations. We have proposed a generic scalability model for analyzing distributed systems in [6]. It takes the view that scalability of distributed systems should be analyzed considering related issues such as consistency, synchronization, and availability. We give below the essence of the model.

$scalability = f(avail, sync, consis, workload, faultload)$

- *avail* is availability—can be quantified as the ratio of the number of transactions accepted versus those submitted.
- *consis* is consistency, itself a function of update ordering and consistency granularity. Update ordering refers to the update ordering mechanisms across replicas of an object and can be one of causal, serializable or PRAM. Consistency granularity refers to the grain size at which consistency needs to be maintained.
- *sync* refers to synchronization among the replicas. The two dimensions of synchronization are how often the replicas are synchronized and the mode of synchronization (push/pull).
- *workload* can be broken down into workload intensity (number of transactions per second or number of clients) and workload service demand characterization (CPU time for operations).
- *faultload* refers to the failure sequences and the number as well as location of the replicas.

The scalability model given above is useful to identify bottlenecks in distributed systems. By applying the scalability model on shared object spaces, we have identified the key bottlenecks that inhibit existing shared object spaces (with the exception of Globe) from scaling up to the Internet:

- Centralized Components
  Many existing DSMs and shared object spaces have some centralized components that affect their scalability. For instance, Orca has a sequencer for realizing totally ordered group communication, while others like T Spaces [4] have a centralized component for object lookups.
- Failures
  Existing shared object spaces do not handle failures. For instance, JavaSpaces and global shared objects do not handle failures of transaction coordinator, while Orca does not handle failure of the sequencer.
- Object Lookup
  Given an object identifier (id), efficient mechanisms must exist that maps the id to the node that either stores a replica or stores meta-data about the replica. Existing shared object spaces such as T Spaces use centralized lookup mechanisms. Object lookup mechanisms in distributed object middleware such as CORBA and DCOM also have difficulty in handling failures and scaling up.
- Consistency
  Several existing DSM systems such as TreadMarks, Munin and shared object spaces such as JavaSpaces provide relaxed consistency mechanisms such as release consistency and entry consistency. Relaxed consistency mechanisms have also been explored in other areas [66, 52]. However, to our knowledge,

these mechanisms have not been evaluated in Internet scale systems. Peer-to-Peer (P2P) systems which have been scaled to the Internet, such as Pastry [69] and Tapestry [17] assume replicas are read-only.

3. SID Techniques for Data Management.

3.1. Computing Grids. Globus [39] a de-facto standard toolkit for grid computing systems, relies on explicit data transfers between clients and computing servers. It uses the GridFTP protocol [19] 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 [51] 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. It does not address consistency issues per se. Further, the RMS is centralized and may not scale up. The other grid paper that has addressed data management issues [29] outlines possible use-cases and gives higher level view of the data management requirements in a grid. The quorum scheme it describes for handling read-write may have to be modified in an Internet kind of an environment to handle quorum dynamics. Further, it does not address various granularities of replication and uses locks for synchronization. The paper [78] also addresses read-write data consistency in a grid environment based on a lazy update propagation algorithm. The update propagation algorithm is based on timestamps and may not scale up to work in a large scale grid environment (Update conflicts are handled manually by application programmer - non-trivial task). Attempts have also been made to extend the existing 2Phase Commit (2PC) based algorithms [82]. These would need global agreement and may be expensive in an Internet setting.

3.2. Data Grids. A generic architecture for handling large data sets in grid computing environments has been proposed in [27]. It describes the way data grid services such as replication and replica selection can be built over basic services of data and meta-data access. It assumes that replicas (file instances) are read-only.

GriPhyN [31] attempts to support large-scale data management in High Energy Physics (HEP) applications as well as for astronomy and gravitational wave physics. GriPhyN provides users transparent access to both raw and processed data (The term virtual data is used to refer to both). It can convert raw data to processed data by scheduling required computations and data transfers. GriPhyN is built on top of Globus. It takes application meta-data and maps it into a Directed Acyclic Graph (DAG), which is an abstract representation of the required actions on data sets. A request planner takes the DAG and transforms it into a concrete DAG, which can be executed by a grid scheduling system such as Condor-G [42].

CERN, the European organization for nuclear research, is also involved in handling computation on large data sets in the HEP area. Object level as well as file level replication for data grids has been explored in [79], a CERN effort. It also assumes files are read only and can be replicated without need for consistency protocols. They support replica catalogs to handle meta-data. Actual file/object transfers are achieved using GridFTP [19].

Data related activities on the grid such as queuing, monitoring and scheduling need to be carefully managed, as data could become bottleneck for data intensive applications. Currently, these data related tasks are performed manually or by simple scripts. The main goal of Stork [85] was to make data a first class citizen on the grid. Data placement jobs have different characteristics from compute intensive jobs and so, may have to be treated differently. Stork is a separate scheduler for scheduling and managing data intensive jobs on grid. Data related activities are represented in the form of a DAG. Stork can interact with higher level planners such as Directed Acyclic Graph Manager (DAGman) which is a part of CondorG. Enhancements have been made to DAGman to make it submit compute intensive jobs to grid schedulers such as CondorG and data intensive jobs to Stork. Stork also supports different heterogeneous storage systems and various data transfer protocols. Case studies have demonstrated the use of Stork as a pipeline between two heterogeneous storage systems and for runtime adaptation of data transfers.

3.3. Content Distribution Networks. Web servers had difficulty in handling the *flash crowd* problem. The *flash crowd* problem refers to a large number of requests coming in suddenly, overwhelming the server's bandwidth, or CPU or back-end transaction infrastructure. Web servers have bursty request nature, for instance during a football match in World Cup or during an election counting process, resulting in the flash crowd problem. Content Distribution Networks (CDNs) such as Akamai [43] have been proposed to handle this problem and to enable web servers to scale up. A separate infrastructure of dedicated servers spread across the Internet was built by several companies to offload content distribution from web servers or to deliver content from the edge of the Internet. Akamai's CDN consists of over twelve thousand servers across thousand different networks. They use either URL rewriting or DNS interposition to redirect client requests to the proximal CDN server.

Studies have shown that caching is beneficial in CDNs as they mainly deliver images or videos (static content) [44]. Akamai CDNs achieved cache hit rates of nearly 88% in another study that compared the CDNs with P2P file sharing systems for distributing content [76]. This shows that CDNs are beneficial for content delivery and can reduce response time for clients. However, another study has shown that the average response time for clients is not affected by employing CDNs [44]. But they avoid worst case of badly performing servers rather than routing client requests to an optimal CDN server.

Cache consistency becomes a challenging issue in order to deliver non-static content to clients. Traditional caching mechanisms such as leasing [22] may not be directly applicable to CDNs. Origin servers would have to keep track of each CDN proxy that caches an object (web document) from the server. It must also manage the lease related issues for that CDN proxy, including notifying the CDN proxy on updates to the object. The CDN proxy has to renew the lease to receive further notifications. Mechanisms for CDNs must be scalable, requiring the CDN proxies to cooperatively maintain consistency. Cooperative leases has been proposed as a scalable mechanism for maintaining cache consistency in CDNs. [12, 11]. Each object is assigned a Δ parameter, which indicates the time or the rate $1/\Delta$ at which an origin server notifies interested CDN proxies of updates to that object. This allows consistency to be relaxed implying that CDN proxy can be notified only once every Δ time units, instead of after every update. Leases are cooperative, meaning that a CDN proxy acts as a leader for a CDN proxy group for lease related interactions with an origin server. The leader is responsible for notifying the other CDN proxies. This reduces both the state maintained at the origin server and the number of updates it must send.

3.4. Data Management in Distributed Mobile Systems. Distributed Mobile Systems (DMS) are distributed systems in which some nodes may be mobile and may have constraints. These constraints could be battery or memory or computing power related. Data could either be stored on or be accessed from mobile devices. Different kinds of management have been identified, with respect to the level of transparency to applications in [54]. Client transparent adaptation allows applications to seamlessly access data without being aware of mobility, with the system providing complete support. The other extreme is a *laisse-faire* model in which adaptation is entirely at user level, with the system providing no support. There are a wealth of strategies between the two extremes, that allow applications to be aware of mobility in varying degrees including application aware adaptation and extended client server models.

Coda [74] was one of the early file systems that allows clients to seamlessly access information, an example of client transparent adaptation. The main goal of Coda was to enable operations to be performed on a shared data repository, even in the face of disconnected operations. Disconnections may be frequent in DMS. *Venus* is the cache manager on each client that manages the cache, hiding mobility from the application. Venus caches volume mappings, with a volume referring to a subtree of the Coda namespace. In the face of connected operations, Coda uses server replication and callback based cache coherence to ensure session semantics (contents will be latest when a session is starting and after it ends) for applications. During disconnections, Venus relies on cache contents and propagates failure to application when a cache miss occurs. When disconnection ends, Coda reverts back to server replication by using reintegration operations using logs.

Application aware adaption has been used in the Odyssey system [21]. Odyssey provides a clean separation between the concerns of the system and the application: system monitors resource dynamics and notifies applications if required, but retains control of resource allocation mechanism; while applications specify mapping of resource levels to *fidelity* levels. Fidelity is defined as the degree to which client data matches with server's. It has multiple dimensions of consistency, frame rate and image quality for video data as well as resolution for spatial data. Building a system that allows diverse fidelity levels necessitates type awareness - client code is responsible for handling particular data types. This is achieved through the use of *wardens*, which are specialized code components that encapsulate system level support at the client. Wardens are subordinate to *Viceroy*, which is responsible for centralized resource management.

Odyssey is an example of client based application aware adaptation. Rover [13] is a system that allows client-server adaptation. This means that some code required for adaption would also reside in server. Rover uses the concept of Relocatable Dynamic Objects (RDOs) for data types handled by the application. The application programmer splits the program containing RDOs into those that reside on the client and those that run on servers. This requires that the adaptation code be resident on origin servers. Another approach has been taken to avoid this, named as proxy based adaptation. The adaptation is done by the proxy, which acts on behalf of clients. The Barwan project [30] is an example. Flexible client server model for application aware adaptation has been proposed in the Bayou system [84]. It allows clients to read/write shared data. Conflicts resolution is handled by using application specific dependency checks and merge procedures. It provides eventual

consistency, an unbounded consistency mechanism that allows replicas to diverge, but be consistent after an unspecified time.

3.5. Software Availability and Usage Summary. Globus is a widely used toolkit and is available as an open source software. Stork is a research prototype, while GriPhyN and CERN have been deployed and used. Akamai's CDNs are widely deployed and used, while cooperative leases [12] is a research prototype. Coda and Odyssey are the distributed mobile systems software that are widely deployed and used.

4. Large Scale Data Management Techniques.

4.1. P2P Data Management. We first give an overview of P2P file sharing systems starting from the initial unstructured P2P systems such as Napster to super-peer systems such as Kazaa before discussing structured P2P systems. We go on to discuss P2P storage management systems such as Oceanstore.

4.1.1. P2P File Sharing Systems. P2P as an area became popular only after the advent of Napster, a file sharing system. Napster [57] was used for sharing music files. Meta-data about files is stored in a global directory, which is stored in a centralized server. The meta-data stored information about music files themselves, which were downloaded from peers. Gnutella [33] came up with a decentralized search protocol for file sharing applications. Gnutella can be seen to be a purely decentralized unstructured P2P system. The term "unstructured" refers to the lack of structure in the overlay, which is mostly a random graph. Search was achieved by flooding the network or by using random walks. Freenet added a mechanism to *route* requests to possible content locations, based on best effort semantics. Freenet also adds a notion of anonymity to the data shared. The main advantage of the unstructured P2P systems was that complex queries could be easily handled. By complex queries, we mean queries such as "get all nodes with processing speed $>$ 3GHz and RAM $>$ 1GB and storage $>$ 100GB". This is because the query is sent to each node and evaluated explicitly. However, deterministic guarantees for searching are difficult to provide in these systems.

Initial attempts at introducing structure to the overlay in P2P systems resulted in super-peer systems, with some nodes (which have better capabilities) acting as super-peers. The other nodes act as clients to the super-peers, which form a P2P overlay among themselves. Super-peers made searching more efficient for complex queries, by exploiting the heterogeneous nature of nodes (some nodes have better capabilities and more importantly, better connectivity than others). An example of a popular super-peer system is Kazaa (`http://www.kazaa.com`). However, handling super-peer failures requires replicating super-peers (otherwise the clients may become disconnected). K-replicas can be created in each cluster, resulting in reduced load on the super-peers [93]. However, this may make replicas client aware. Other design issues in super-peer systems include cluster size and dynamic layer management. A large cluster size is good for aggregate bandwidth, but may create bottlenecks. A small cluster size avoids bottlenecks, but may reduce search efficiency. Dynamic layer management allows nodes to play super-peer or client nodes adaptively, thereby making the super-peer network more efficient [95].

The third generation of P2P systems introduced structure in the overlay network. The motivation came from providing deterministic search guarantees, partitioning the load over the available machines effectively, scaling to large numbers and achieving fault-tolerance. The Distributed Hash Table (DHT) was mainly used as the structure for overlay formation. It was based on the Plaxton data structure [23]. Nodes are given identifiers (ids) from an id space. Application objects are also given ids from the same space. The DHT provides a mapping from the application object id (key) to the node id that is responsible for that key. Each node has a routing table consisting of neighbours and performs routing functions to lookup objects. Various DHTs have been proposed, each having different routing algorithms and routing table maintenance. Geometric interpretations of DHTs have been given in [45] (but the focus of that paper was mainly to study the static resilience of DHTs). Chord [40] is based on a ring, while Content Addressable Network (CAN) is based on a hypercube, Plaxton data structure is based on a tree, while Pastry [69] is a hybrid geometry combining the tree and the ring. We discuss some of these structured P2P systems in more detail below.

Chord provides the lookup abstraction of DHTs through the method: lookup(key) which maps a key to a node responsible for it. Chord uses consistent hashing to assign m-bit identifiers to both Chord nodes and application objects. The ids are arranged in a ring fashion (modulo 2^m). A key k maps to the first node whose id is equal to or follows k in the identifier space (this node is known as successor(k)). Each node maintains a pointer to its successor in the ring. Routing proceeds along the ring till a key is straddled between two node ids, with the second node id being the destination. Each node also maintains information on $O(\log(N))$ (for N nodes) other nodes in the form of a *finger table* in order to speed up routing. Even if nodes in the finger

table were to fail, only efficiency is affected, but not correctness. As long as each node is able to connect to its successor, routing is guaranteed to finish in $O(\log(N))$ time.

CAN routes over a hypercube. Each CAN node stores a chunk (or zone) of the hash table. Each node also stores information on adjacent zones in the table. This is again to speed up routing. Lookup requests for a particular key are routed towards a CAN node whose zone contains that key. Requests are routed by correcting bits (n bits for a n-dimensional hypercube). Generally tree based DHTs such as the Plaxton data structure allow bits to be corrected in order (from MSB to LSB of key), while hypercube based DHTs allow bit correction in any order. This makes routing more resilient to node/link failures.

Pastry can be viewed as having a hybrid geometry due to its use of tree based routing and ring like neighbour formation. It provides a *route* abstraction to applications. The *route(msg, key)* ensures that the message with a given id is routed to a node with the closest matching id as key among all live nodes. Each node keeps track of its immediate neighbours in the node id space by maintaining leaf sets. They also store information about a few other nodes that have prefix matching ids in the form of a routing table. Pastry takes into account network locality in routing. This means that a given message will be routed to the nearest node that is alive and that has the closest matching id as the key. Routing takes place by prefix matching, with each hop taking the message one bit closer in the node id space, resulting in $O(\log(N))$ hops.

4.1.2. P2P File Storage Systems. Ivy [56] is a read/write P2P file system that provides an NFS-like abstraction for programmers. Ivy provides NFS-like semantics in a failure free environment. Under network partitions and failures, Ivy uses logs to allow applications to detect and resolve conflicts. Ivy logs are specific to each participant and host. The logs are stored in DHash, a DHT based P2P block storage system over which Ivy is built. Participants can *read* other logs, but write only his/her log while updating the file system. Ivy uses versioning vectors to detect conflicting updates and provides information to application level conflict resolvers. Ivy system demonstrated a performance within 2-3 factor of NFS performance in a WAN testbed.

PAST [15] is an Internet based P2P storage utility. It offers persistent storage services, availability, security and scalability. PAST provides *insert*, *reclaim* and *retrieve* operations on files. Since a file cannot be inserted multiple times, files are assumed to be immutable in PAST. It must be noted that PAST is an extension of Pastry to provide a file storage system. On insertion of a file into PAST, the file is routed by Pastry to k-nodes with closest matching ids as the file id and that are alive. The set k will be diverse with respect to location, capabilities and connectivity due to the randomization of the identifier space. File availability is ensured as long as all k nodes do not fail simultaneously. It provides security using optional smartcards that are based on a public-key cryptosystem.

Oceanstore [49] is an Internet based file system that provides persistence and availability of files by using a two-tiered system. The upper tier consists of capable machines with good connectivity. These machines act as an *inner* circle of servers for serializing updates. The lower tier consists of less capable machines which only provide storage resources to the system. Pond [67] is an Oceanstore realization that provides fault tolerant durable storage to applications. It uses erasure coding to store data. Erasure coding [20] is a technique that allows a block to be split into m fragments, which are encoded into n fragments ($n > m$). The key property of erasure coding is that it ensures that the block can be reconstructed from any m of the n coded fragments. Oceanstore uses Tapestry [17], another DHT, to store the erasure coded fragments (based on fragment number + block id). Oceanstore uses primary copy replication to ensure consistency of file blocks. It handles read/write data by a versioning mechanism in which any write operation creates a new version of the data. The problem is then reduced to one of finding the most recent version of the file.

4.1.3. Observations. Ivy has the disadvantage that it leaves write conflict resolution to the application, limiting the scalability. PAST provides a persistent caching and storage management layer on top of Pastry. It provides *insert*, *lookup* and *reclaim* operations on files. However, it also assumes files are immutable, as files cannot be inserted multiple times with the same id. Oceanstore's versioning mechanism has not been proved scalable. The evaluations on Oceanstore and Pond [67] have not considered conflicting write operations and have assumed there is a single write per data block. Moreover, Oceanstore assumes an inner circle of reliable servers to ensure consistency. Further, all the three storage systems (Ivy, PAST and Oceanstore) have been built over DHTs. DHTs provide support for only limited queries (exact matching kind) and may not allow application specific criterion for data placement. In the words of [47], virtualization (through DHTs) "destroys locality and application specific information". However, there have been recent efforts that enable DHTs to handle advanced queries such as those handled in [18].

4.2. P2P Extensions to DDBMS. A simplistic view of a traditional distributed database management system is that it uses a centralized server to provide a global schema and ACID properties through transactions. Several approaches have extended these techniques to work in a decentralized manner, to apply to Internet or P2P systems. Active XML [9] provides dynamic XML documents over web services for distributed data integration. It is a model for replicating (whole file) and distributing (parts of a file) XML documents by introducing location aware queries in X-Path and X-Query. It also provides a framework by which peers perform decentralized query processing in the presence of distribution and replication. It allows peers to optimize localized query evaluation costs, by a series of replication steps.

Edutella [58] attempts to design and implement a schema based P2P infrastructure for the semantic web. It uses W3C standards RDF and RDF Schema as the schema language to annotate resources on the web. It uses RDF-QEL as an expressive query exchange language to retrieve the data stored in the P2P network. It uses super-peer routing indices that include schema and other index information.

Piazza [83] is a peer data management system that facilitates decentralized sharing of heterogeneous data. Each peer contributes schemas, mappings, data and/or computation. Piazza provides query answering capabilities over a distributed collection of local schemas and pairwise mappings between them. It essentially provides a schema mediation mechanism for data integration over a P2P system.

P2P Information Exchange and Retrieval (PIER) [38] is a P2P query engine for query processing in Internet scale distributed systems. PIER provides a mechanism for scalable sharing and querying of finger print information, used in network monitoring applications such as intrusion detection. It provides best effort results, as achieving ACID properties may be difficult in Internet scale systems. The query engine does not assume data is loaded into databases on all peers, but is available in their *natural habitats* in file systems. PIER is realized over CAN, the hypercube based P2P system.

PeerDB [60] is an object management system that provides sophisticated searching capabilities. PeerDB is realized over BestPeer [59], which provides P2P enabling technologies. PeerDB can be viewed as a network of local databases on peers. It allows data sharing without a global schema by using meta-data for each relation and attributes. The query proceeds in two phases: in the first phase, relations that match the user's search are returned by searching on neighbours. After the user selects the desired relations, the second phase begins, where queries are directed to nodes containing the selected relations. Mobile agents are dispatched to perform the queries in both phases.

4.3. Software Availability and Usage Summary. Gnutella and Napster have been widely deployed and used. Chord is a research prototype that is also available as an open source software. Pastry is also available as an open source software and has also been used widely. CAN and Ivy are research prototypes about which deployment information is not available. PAST and Oceanstore are research prototypes that have been deployed and used in the Planetlab testbed.

Edutella is available as an open source software. The authors do not have information on the deployment/availability on other research prototypes Piazza, PeerDB and Active XML. PIER has been deployed in the Planetlab testbed.

5. State of the Art Data Management.

5.1. SID Techniques: State of the Art.

5.1.1. P2P Techniques in Grids. JuxMem [2] provides a data sharing service for grids by integrating DSM concepts with P2P systems. It is realized over (Juxtapose) JXTA [34], an emerging framework for developing P2P applications. JuxMem uses cluster advertisements to advertise the amount of memory each peer can provide to the global storage. It is organized into a federation of clusters, with each cluster having a Cluster Manager (CM). The CM is responsible for storing all cluster advertisements in its group. The CMs across clusters form a DHT. Actually, the amount of memory provided in the cluster advertisement is hashed and the CM with the closest matching id in the DHT stores this advertisement. When a client asks for a block of memory with a given rounded size (fixed sized blocks can only be supported), the size is hashed and the cluster advertisement which provides that size is retrieved from the CM with the closest matching id. The cluster advertisement has the details of the actual storage provider. Recent extensions to JuxMem [14] provide mechanisms to decouple consistency protocols from fault-tolerance mechanisms. This allows the use of standard DSM consistency protocols to integrate fault-tolerance components. In particular, DSM consistency schemes such as home based consistency [41] which assume a single home node for serializing concurrent writes, can be made fault-tolerant by having a group of nodes as the home node. This requires group membership protocols, as

well as an atomic multicast protocol, which is achieved by using consensus protocols based on Failure Detectors (FDs) [26]. The data sharing mechanisms of JuxMem have only been evaluated at the cluster level.

The replica location problem has been addressed in grids using P2P concepts in [5]. It proposes a P2P realization of the Replica Location Service (RLS), a key component of data grids. The Logical File Name (LFN) is hashed to give the identifier for a replica. The node with the closest matching id as the LFN hash contains the LFN to Physical File Name (PFN) mapping. This is the meta-data stored in RLS for file lookup. It also proposes an update protocol to handle consistency of meta-data. The RLS realization is based on Kademlia [63]. Kademlia is a structured P2P system that uses a novel XOR metric for routing—distance between two nodes is defined as the eXclusive OR (XOR) of their numeric ids. A Kademlia node forms log(n) neighbours, where neighbour i is at XOR distance $[2^i, 2^{i+1}]$. The neighbour set is same as that formed by a tree based DHT PRR [23]. Even the failure-free routing in Kademlia is similar to PRR, in that bits are corrected from left to right. However, in the case of failures, XOR metric allows bits to be corrected in any order. This implies that the static resilience[2] of Kademlia is better compared to PRR [45].

5.1.2. Replica Placement in CDNs. Optimal placement of replicas in CDNs is a non-trivial task and has not been addressed. QoS aware replica placement was proposed in [92] to meet QoS requirements of clients with the objective of minimizing the replication cost. The replication cost includes cost of storage and consistency management, while QoS is specified in terms of distance metrics such as hop count. Two problems are formulated: Replica-aware and Replica-blind. In replica-aware model, the CDN servers are aware of where object replicas are stored in the CDN network. This helps the servers to redirect client requests to the nearest replica. In the replica blind model, application or network level routing ensures client requests are routed to CDN servers, with servers being transparent to replica location. Each replica (CDN server) serves requests coming to it. Dynamic programming techniques are used to arrive at near optimal solutions for the optimal replica placement problem, which is shown to be NP-complete.

5.1.3. Distributed Mobile Storage System. Segank [80] provides an abstraction of a shared storage system for heterogeneous storage elements. The motivation was that traditional mechanisms for managing data in distributed mobile environments such as Coda and Bayou, have time consuming merge operations. In Coda, updates are released to the server before becoming visible on clients. If servers are physically far away, this could increase the time after which updates become visible. Bayou uses full replication, leading to potentially expensive merge operations. Segank handles data location problem when data could be located on any subset of devices, by using a location and topology sensitive multicast-like (named as segankcast) operation. It allows lazy P2P propagation of invalidation information to handle consistency of replicated data. It also uses a distributed snapshot mechanism to ensure a consistent image across all devices for backup. It must be observed that Segank uses only unstructured P2P system concepts. This implies that Segank cannot provide deterministic search guarantees.

5.2. Large Scale Data Management: State of the Art. We shall explain the current state of the art in P2P data management along four directions: integrating structured and unstructured P2P systems providing Quality of Service (QoS) guarantees in P2P systems, composable consistency for P2P systems and large scale DHT deployment. We also explain the state of the art in P2P DBMS.

5.2.1. Integrating Structured and Unstructured P2P Systems. An attempt has been made in [55] to improve structured P2P systems along three directions where they were traditionally known to perform worse compared to unstructured P2P systems: handling churn, exploiting heterogeneity and handling complex queries. In P2P systems, node/network dynamics resulting in routing-table updates and/or data movement is known as churn. The paper [55] shows that MS Pastry, an implementation of Pastry, can handle churn well by using a periodic routing table maintenance protocol. This protocol updates failed routing table entries. It also has a passive routing table repair protocol. They demonstrate that by exploiting structure, MS Pastry can handle churn better than unstructured P2P systems. Heterogeneity is difficult to handle in structured P2P systems due to constraints on data placement and neighbour selection. MS Pastry handles heterogeneity in two ways: one by using super-peer concepts; second, by modifying neighbour selection to handle capacity. MS Pastry is also extended to handle complex queries by introducing new techniques for flooding or random walks. Flooding is achieved by sending the message to all nodes in the routing table. Random walk is achieved by using a tag containing the set of nodes to visit, a queue of nodes in the routing table row and a bound on number of rows to traverse. A few other efforts have also been made recently to make structured P2P systems handle

[2]Static Resilience measures the goodness of a DHT routing algorithm before recovery mechanisms take effect

range queries [16], multi-dimensional queries [65] as well a query algebra [73]. A Scalable Wide Area Resource Discovery (SWORD) [62] has been built to realize resource discovery over WANs by supporting multi-attribute range queries over DHTs.

Another approach to integrate structured and unstructured P2P systems has been made in the Vishwa computing grid middleware [53]. Vishwa uses the task management layer to handle initial task deployment and load adaptability of the tasks. The task management layer is realized using unstructured P2P concepts and allows capability based resource clustering. The reconfiguration layer of Vishwa is realized as a structured P2P layer and stores information needed to handle node/network failures. The two layered architecture has also been used for data management in Virat [1, 7]. Virat provides a shared object space abstraction over a wide-area distributed system. Virat has been extended to a replica management middleware for P2P systems [8]. The unstructured layer forms neighbours based on node capabilities (in terms of processing power, memory available, storage capacity and load conditions). A structured DHT is built over this unstructured layer by using the concept of virtual nodes. Virat achieves dynamic replica placement on nodes with given capabilities, which would be very useful in computing/data grids. Detailed performance comparison is also made with a replica mechanism realized over OpenDHT [68], a state of the art structured P2P system. It has been demonstrated that the 99th percentile response time for Virat does not exceed 600 ms, whereas for OpenDHT, it goes beyond 2000 ms in an Internet testbed.

5.2.2. Composable Consistency for P2P Systems. A flexible consistency model known as composable consistency suitable for a variety of P2P applications has been proposed in [72]. The authors have initially surveyed consistency requirements for P2P applications such as personal file access, real time collaboration and database or directory services. The survey showed that different applications need different semantics for read/write and for replica divergence. The main contribution of [72] is the classification of consistency requirements along five orthogonal dimensions: concurrency—degree of conflicting read/write access; replica synchronization—degree of replica divergence; failure handling—data access semantics in the presence of inaccessible replicas; update visibility - time after which local updates may be made globally visible; view isolation—time after which remote updates must be made locally visible. A rich collection of consistency semantics for shared data can be *composed* by combining the above five options. Performance studies have shown that composable consistency in the Swarm system outperforms CoDA [74] in a file sharing scenario, while for a replicated BerkeleyDB database, it provides different consistency mechanisms from strong to time-based.

5.2.3. Providing QoS Guarantees in P2P Systems. Guaranteeing Quality of Service (QoS) parameters such as response time or throughput in P2P systems is a challenging task. An initial attempt was made in [70] at using P2P system concepts for Domain Name System (DNS), which requires efficient data location. It showed that though P2P DNS could provide better fault-tolerance than conventional DNS, lookup performance of O(log(N)) provided by DHTs was far worse compared to conventional DNS. Cooperative DNS (CoDoNS) [89] was proposed to tackle three problems of conventional DNS: susceptibility to Denial of Service (DoS) attacks; lookup delays, especially for flash crowds; lack of cache coherency, preventing quick service relocation in emergencies. CoDoNS has been proposed as a backward compatible replacement for conventional DNS. It provides O(1) lookup time by using the proactive caching layer of Beehive [88]. Beehive enables DHTs to achieve O(1) lookup performance by proactive replication. Traditionally, prefix matching DHTs store an application object at the closest matching node, with each routing step successively matching prefixes, resulting in O(log(N)) lookup performance. By aggressively caching the object all along the lookup path, Beehive achieves O(1) lookup performance for that object. Since, Beehive associates different replication levels for different application objects, an average lookup performance of O(1) is achieved. CoDoNS builds a DNS based on a self-organizing P2P overlay formed across organizations (if each organization can provide a server for CoDoNS). CoDoNS associates a domain name with the node having the closest matching id as the domain name's hashed id. If the home node fails, the node with the next best matching id takes over as the home node for that particular domain. Performance studies over PlanetLab testbed show that CoDoNS achieves lower lookup latencies, can handle slashdot effects and can quickly disseminate updates. However, the use of DHTs as the basis leaves CoDoNS vulnerable to network partitions. For example, if an organization is partitioned from the outside world, while conventional DNS would ensure that local lookups worked correctly, with CoDoNS even local lookups may fail (DHT lookup may go outside the local network even for local lookups—stretch property of DHTs). This suggests that SkipNets [35] may be a better choice for realizing DNS than DHTs. This is because data in SkipNets is organized by using string names which guarantees routing locality. This is in addition to the normal numeric identifier based organization used in DHTs.

5.2.4. Large Scale Deployment. OpenDHT [68] is a public large scale DHT deployment that allows clients to use DHTs without having to deploy them. It provides a shared storage space abstraction using the *get* and *put* primitives. The main motivation for OpenDHT is that it is hard to deploy long running distributed system services, especially in the public domain. OpenDHT is deployed on PlanetLab (http://www.planet-lab.org/), a global testbed for deploying planetary scale services. OpenDHT is deployed on *infrastructure* nodes which alone participate in DHT routing and storage. Clients only *use* the storage space through the *get* and *put* interface on gateway (infrastructure) nodes. OpenDHT allows different mutually untrusting applications to share the DHT. It ensures that clients get a fair share of storage resources without imposing arbitrary quotas—a trade-off between fairness and flexibility. This is achieved by associating a Time-to-Live (TTL) with application objects and letting them expire if clients do not renew them. OpenDHT provides *storage* abstraction of DHTs in contrast to the *lookup* abstraction of Chord or the *routing* abstraction of Pastry.

It is realized over Bamboo DHT(bamboo-dht.org), that is similar to Pastry but has differences in handling node dynamics. OpenDHT is not a shared object space. The level of abstraction provided to programmer is different. For instance, the programmer has to take care of object serialization, RTTI (runtime type inferencing) etc. to realize an object storage on top of the byte storage that OpenDHT provides. OpenDHT provides limited consistency for the shared byte space. Conflict resolution (for concurrent writes) is left to the application, similar to the Bayou system that ensures "eventual consistency", a very loose form of consistency. But conflict resolution is a non-trivial task for the application programmer. 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 [71].

5.2.5. State of the Art P2P DDBMS. Atlas P2P Architecture (APPA) [86] is the current state of the art data management solution for large scale P2P systems. It uses a three layered architecture, with the P2P network forming the lowest layer. This layer could be realized using unstructured or structured or super-peer based P2P concepts. Above this layer, the basic P2P services layer is built. This provides P2P data sharing and retrieving (key based) in the P2P network, support for peer communication, support for peer dynamics (join and leave) and group membership management. Over the basic services layer advanced P2P data management services such as schema management, replication, query processing and security are built. The shared data is in XML format and queries expressed in X-Queries in order to make use of web services. It is realized over JXTA. It provides replica management by extending traditional centralized log based reconciliation techniques for P2P systems. It assumes the existence of a shared storage space for distributed reconciliation by peers. This requires consensus protocols for realization and may be expensive. It has not been evaluated in large scale systems.

A recent effort has been made to provide a middleware based data replication scheme in [94] by using *Snapshot Isolation* (SI) as the isolation level. In SI based DBMS, read operations of a transaction T are handled from a snapshot of the database (set of committed transactions when T started). This implies read operations never conflict with write operations and only write-write conflicts can occur, resulting in more concurrency and consequently better performance. It has been proposed at the cluster level and may not be applicable for P2P systems due to its strong assumption of a totally ordered multicast.

5.3. Software Availability and Usage Summary. Juxmem and Segank are research prototypes. Deployment information on Structella is not available. Vishwa and Virat are research prototypes that are available as open binaries. OpenDHT has been deployed on the Planetlab testbed and is also available as an open source software. APPA is a research prototype.

6. Conclusions. We have presented a scalability taxonomy of data management solutions in distributed systems. We group data management work done in DSMs and shared object spaces in the Centralized/Naively Distributed (CND) data management category. The Sophisticated/Intermediate Data (SID) management techniques include data management in grid computing systems and data grids as well as Content Distribution Networks (CDNs) and data management in distributed mobile systems. These solutions scale better than CND techniques by using distributed data management, instead of centralized approaches. They however, assume an inner set of reliable servers which take care of consistency and reliability issues. However, in order to take the data management services to the edges of the Internet, Large Scale Data (LSD) management techniques make use of P2P concepts. They consequently provide better scalability and fault-tolerance, but at the cost of relaxing consistency (most approaches provide probabilistic guarantees or eventual consistency).

The taxonomy is depicted in figure 6.1. The figure shows the state of the art efforts in orange color and the possible future directions also in blue. The future directions are detailed below.

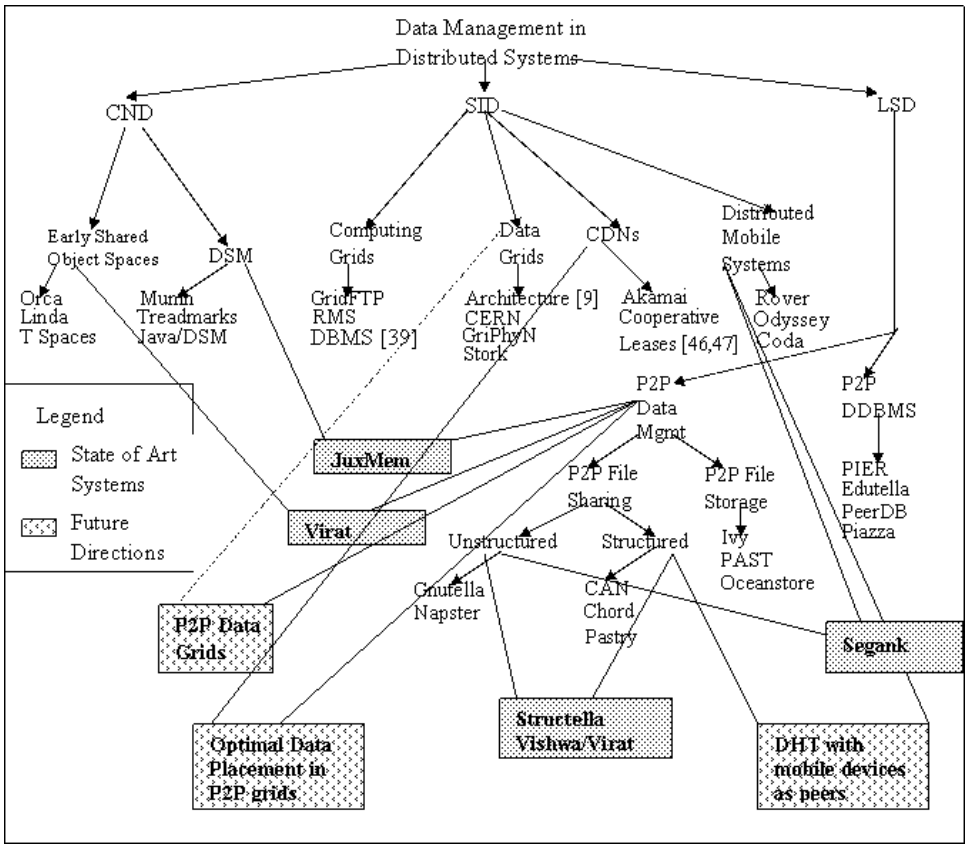

FIG. 6.1. *Pictorial Representation of Scalability Taxonomy*

It can be observed that LSD techniques such as Virat [8] handle large number of small data objects. The case of handling large number of large data objects arises when existing data grids become purely P2P, instead of using SID techniques. The existing LSD techniques may not work in this case, as the size of data objects calls for special mechanisms to handle some operations including updates. Incremental updates or function shipping in combination with LSD data management techniques may have to be explored.

Another interesting avenue for exploration is the use of LSD techniques combined with node mobility. The solutions which have been proposed for handling data management in distributed mobile systems do not use P2P concepts, but assume the presence of reliable servers that handle mobile client requests. When mobile nodes form the P2P overlay, *churn* could be very high due to node mobility. This, coupled with the device constraints, may open up a wealth of research questions.

Optimal data placement techniques which have been proposed for CDNs [92] can be used in P2P grids. Existing data management techniques in grids (or even P2P grids such as P-Grid [46]) do not address optimal replica placement issues. The work [8] provides heuristics for replica placement in P2P grids. But placement of replicas may not be exactly optimal. Thus, we see that techniques for data management in one category can be applied to others to open up research in large scale data management.

REFERENCES

[1] A VIJAY SRINIVAS, M VENKATESHWARA REDDY, AND D JANAKIRAM, *Designing a Replication Service for Large Peer-to-Peer Data Grids*, IEEE Distributed Systems Online, 7 (2006).

[2] GABRIEL ANTONIU, LUC BOUGÉ, AND MATHIEU JAN, *JUXMEM: An Adaptive Supportive Platform for Data Sharing on the Grid*, Scalable Computing: Practice and Experience, 6 (2005), pp. 45–55.

[3] MAARTEN VAN STEEN AND PHILIP HOMBURG AND ANDREW S. TANENBAUM, *Globe: A Wide-Area Distributed System*, IEEE Concurrency, 7 (1999), pp. 70–78.

[4] P WYCKOFF, S W MCLAUGHRY, T J LEHMAN, AND D A FORD, *T Spaces*, IBM Systems Journal, 37 (1998), pp. 454–474.

[5] A. CHAZAPIS, A. ZISSIMOS, AND N. KOZIRIS, *A Peer-to-Peer Replica Management Service for High-Throughput Grids*, in Proceedings of the International Conference on Parallel Processing (ICPP), Washington, DC, USA, June 2005, IEEE Computer Society, pp. 443–451.

[6] A Vijay Srinivas and D Janakiram, *A Model for Characterizing the Scalability of Distributed Systems*, ACM SIGOPS Operating Systems Review, 39 (2005), pp. 64–72.
[7] A Vijay Srinivas and D Janakiram, *A Peer-to-Peer Framework for Collaborative Data Sharing Over the Internet*, Tech. Report IITM-CSE-DOS-2005-28, accepted for publication in IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollobarateCom 2006), IEEE Computer Society Press.
[8] A Vijay Srinivas and D Janakiram, *Node Capability Aware Replica Management for Peer-to-Peer Grids*, Technical Report IITM-CSE-DOS-2006-04, Distributed & Object Systems Lab, Indian Institute of Technology, Communicated to IEEE Transactions on Software Engineering.
[9] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo, *Dynamic XML documents with distribution and replication*, in SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, New York, NY, USA, 2003, ACM Press, pp. 527–538.
[10] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, *TreadMarks: Shared Memory Computing on Networks of Workstations*, IEEE Computer, 29 (1996), pp. 18–28.
[11] Anoop George Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari, *Scalable Consistency Maintenance in Content Distribution Networks Using Cooperative Leases*, IEEE Transactions on Knowledge and Data Engineering, 15 (2003), pp. 813–828.
[12] Anoop Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari, *Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks*, in WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, 2002, ACM Press, pp. 1–12.
[13] Anthony D Joseph, Joshua A Tauber, and M Frans Kaashoek, *Mobile Computing with the Rover Toolkit*, IEEE Transactions on Computers, 46 (1997), pp. 337–352.
[14] G. Antoniu, J.-F. Deverge, and S. Monnet, *How to Bring Together Fault Tolerance and Data Consistency to Enable Grid Data Sharing*, Concurrency and Computation: Practice and Experience, 17 (2006). To appear.
[15] Antony Rowstron and Peter Druschel, *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*, in SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, New York, NY, USA, 2001, ACM Press, pp. 188–201.
[16] Artur Andrzejak and Zhichen Xu, *Scalable, Efficient Range Queries for Grid Information Services*, in P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing, Washington, DC, USA, 2002, IEEE Computer Society, pp. 33–40.
[17] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, *Tapestry: A Resilient Global-Scale Overlay for Service Deployment*, IEEE Journal on Selected Areas in Communications, 22 (2004), pp. 41–53.
[18] D. Bauer, P. Hurley, R. Pletka, and M. Waldvogel, *Bringing efficient advanced queries to distributed hash tables*, in LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04), Washington, DC, USA, 2004, IEEE Computer Society, pp. 6–14.
[19] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke, *Data Management and Transfer in High-Performance Computational Grid Environments*, Parallel Computing, 28 (2002), pp. 749–771.
[20] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, *An xor-based erasure-resilient coding scheme*, 1995.
[21] Brian D Noble, M Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker, *Agile Application-Aware Adaptation for Mobility*, in SOSP '97: Proceedings of the sixteenth ACM symposium on Operating Systems Principles, New York, NY, USA, 1997, ACM Press, pp. 276–287.
[22] C Gray and D Cheriton, *Leases: an Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency*, in SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles, New York, NY, USA, 1989, ACM Press, pp. 202–210.
[23] C Greg Plaxton, Rajmohan Rajaraman, and Andrea W Richa, *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*, in SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel Algorithms and Architectures, New York, NY, USA, 1997, ACM Press, pp. 311–320.
[24] N. Carriero and D. Gelenter, *Linda in Context*, Communications of the ACM, 4 (1989), pp. 444–458.
[25] J. B. Carter, *Design of the Munin Distributed Shared Memory System*, Journal of Parallel and Distributed Computing, 29 (1995), pp. 219–227.
[26] T. D. Chandra and S. Toueg, *Unreliable Failure Detectors for Reliable Distributed Systems*, Journal of the ACM, 43 (1996), pp. 225–267.
[27] Chervenak, A, Foster, I, Kesselman, C, Salisbury, C, and Tuecke, S, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*, Journal of Network and Computer Applications, 23 (2001), pp. 187–200.
[28] U. Dayal, K. Ramamritham, and T. M. Vijayaraman, eds., *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, IEEE Computer Society, 2003.
[29] Dirk DÃijllmann and Ben Segal, *Models for Replica Synchronisation and Consistency in a Data Grid*, in HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), Washington, DC, USA, 2001, IEEE Computer Society, p. 67.
[30] Eric A Brewer, Randy H Katz, Elan Amir1, Hari Balakrishnan, Yatin Chawathe, Armando Fox, Steven D Gribble, Todd Hodes, Giao Nguyen, Venkata N Padmanabhan, Mark Stemm, Srinivasan Seshan, Tom Henderson, Joshua A Tauber, and M Frans Kaashoek, *A Network Architecture for Heterogeneous Mobile Computing*, IEEE Personal Communications, 5 (1998), pp. 8–24.
[31] Ewa Deelman, Carl Kesselman, Gaurang Mehta, Leila Meshkat, Laura Pearlman, Kent Blackburn, Phil Ehrens, Albert Lazzarini, Roy Williams, and Scott Koranda, *GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists*, in Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02), Washington, DC, USA, 2002, IEEE Computer Society, p. 225.
[32] A. Finkelstein, C. Gryce, and J. Lewis-Bowen, *Relating Requirements and Architectures: A Study of Data-Grids*,

Journal of Grid Computing, 2 (2004), pp. 207–222.
[33] Gnutella, *The Gnutella protocol specification v0.4.* http://www9.limewire.com/developer/gnutella protocol 0.4.pdf 2000.
[34] L. Gong, *JXTA: A Network Programming Environment*, IEEE Internet Computing, 5 (2001), pp. 88–95.
[35] Harvey, Nicholas J. A., Jones, Michael B., Saroiu, Stefan, Theimer, Marvin, and Wolman, Alec, *Skipnet: A scalable overlay network with practical locality properties*, in Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, United States, March 2003, USENIX Association.
[36] Henri E Bal, M Frans Kaashoek, and Andrew S Tanenbaum, *Orca: A Language for Parallel Programming of Distributed Systems*, IEEE Transactions on Software Engineering, 18 (1992), pp. 190–205.
[37] Henri E Bal, Raoul Bhoedjang, Rutger Hofman, Ceriel Jacobs, Koen Langendoen, Tim Ruhl, and M Frans Kaashoek, *Performance evaluation of the orca shared-object system*, ACM Transactions on Computer Systems, 16 (1998), pp. 1–40.
[38] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica, *Querying the Internet with PIER.*, in VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany, J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, eds., Morgan Kaufmann, 2003, pp. 321–332.
[39] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Intl Journal of Supercomputer Applications, 11 (1997), pp. 115–128.
[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*, IEEE/ACM Transactions on Networking, 11 (2003), pp. 17–32.
[41] L. Iftode, J. P. Singh, and K. Li, *Scope Consistency: a Bridge Between Release Consistency and Entry Consistency*, in SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, 1996, ACM Press, pp. 277–287.
[42] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, in HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), Washington, DC, USA, 2001, IEEE Computer Society, p. 55.
[43] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl, *Globally Distributed Content Delivery*, IEEE Internet Computing, 06 (2002), pp. 50–58.
[44] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek, *The measured performance of content distribution networks*, Computer Communications, 24 (2001), pp. 202–206.
[45] K Gummadi, R Gummadi, S Gribble, S Ratnasamy, S Shenker, and I. Stoica, *The Impact of DHT Routing Geometry on Resilience and Proximity*, in SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2003, ACM Press, pp. 381–394.
[46] Karl Aberer, Philippe Cudre-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt, *P-Grid: a Self-Organizing Structured P2P System*, ACM SIGMOD Record, 32 (2003), pp. 29–33.
[47] P. J. Keleher, B. Bhattacharjee, and B. D. Silaghi, *Are Virtualized Overlay Networks Too Much of a Good Thing?*, in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002, Springer-Verlag, pp. 225–231.
[48] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy, *Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors*, in ISCA '90: Proceedings of the 17th annual international symposium on Computer Architecture, New York, NY, USA, 1990, ACM Press, pp. 15–26.
[49] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, *OceanStore: an Architecture for Global-Scale Persistent Storage*, SIGARCH Computer Architecture News, 28 (2000), pp. 190–201.
[50] L G Alex Sung, Nabeel Ahmed, R. andHerman Li, Mohamed Ali Soliman, and David Hadaller, *A Survey of Data Management in Peer-to-Peer Systems.* CS856 Web Data Management, 2005. School of Computer Science, University of Waterloo.
[51] L Guy, P Kunszt, E Laure, H Stockinger, and K Stockinger, *Replica Management in Data Grids.* Technical Report, GGF Working Draft, 2002.
[52] M. Ahamad and R. Kordale, *Scalable Consistency Protocols for Distributed Services*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 888–903.
[53] M. V. Reddy, A. V. Srinivas, T. Gopinath, and D. Janakiram, *Vishwa: A Reconfigurable Peer-to-Peer Middleware for Grid Computing*, in 35th International Conference on Parallel Processing, IEEE Computer Society Press, 2006, pp. 381–390.
[54] Mahadev Satyanarayanan, *Accessing Information on Demand at any Location. Mobile Information Access*, IEEE Personal Communications, 3 (1996), pp. 26–33.
[55] Miguel Castro, Manuel Costa, and Antony Rowstron, *Debunking Some Myths About Structured and Unstructured Overlays*, in Proceedings of the 2nd Usenix Symposium on Networked System Design and Implementation, Boston,MA, May 2005.
[56] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, *Ivy: a Read/Write Peer-to-Peer File System*, SIGOPS Operating Systems Review, 36 (2002), pp. 31–44.
[57] NAPSTER, *Napster media sharing system.* http://www.napster.com
[58] W. Nejdl, W. Siberski, and M. Sintek, *Design issues and challenges for RDF- and schema-based peer-to-peer systems*, SIGMOD Record, 32 (2003), pp. 41–46.
[59] W. S. Ng, B. C. Ooi, and K.-L. Tan, *BestPeer: A Self-Configurable Peer-to-Peer System.*, in Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA, IEEE Computer Society, 2002, p. 272.
[60] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou, *PeerDB: A P2P-based System for Distributed Data Sharing.*, in Dayal

et al. [28], pp. 633–644.
[61] Object Management Group, *The Common Object Request Broker: Architecture and Specification*. 2. 3. 1, October 1999.
[62] Oppenheimer, D., Albrecht, J., Patterson, D., and Vahdat, A., *Design and Implementation Tradeoffs for Wide-area Resource Discovery*, in Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005. HPDC-14, Washington, DC, USA, July 2005, IEEE Computer Society, pp. 113–124.
[63] Petar Maymounkov and David Mazires, *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*, in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002, Springer-Verlag, pp. 53–65.
[64] Peter J Keleher, *The Relative Importance of Concurrent Writers and Weak Consistency Models*, in ICDCS '96: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96), Washington, DC, USA, 1996, IEEE Computer Society, p. 91.
[65] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina, *One Torus to Rule Them All: Multi-Dimensional Queries in P2P Systems*, in WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, New York, NY, USA, 2004, ACM Press, pp. 19–24.
[66] M. Raynal, G. Rhia-kime, and M. Ahamad, *Serializable to Causal Transactions for Collaborative Applications*, in Proceedings of the 23rd Euromicro Conference, Budapest, Hungary, September 1997.
[67] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, *Pond: The OceanStore Prototype*, in Proceedings of the Conference on File and Storage Technologies, USENIX Association, 2003.
[68] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, *OpenDHT: a public DHT service and its uses*, in SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2005, ACM Press, pp. 73–84.
[69] A. Rowstron and P. Druschel, *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*, in Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Midleware 2001), Heidelberg, Germany, November 2001, pp. 329–350.
[70] Russ Cox, Athicha Muthitacharoen, and Robert Morris, *Serving DNS Using a Peer-to-Peer Lookup Service*, in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002, Springer-Verlag, pp. 155–165.
[71] S. Rhea, B. G. Chun, J. Kubiatowicz, and S. Shenker, *Fixing the Embarrassing Slowness of OpenDHT on PlanetLab*, in Proceedings of USENIX WORLDS 2005, USENIX Association, 2005.
[72] Sai Susarla and John Carter, *Flexible Consistency for Wide area Peer Replication*, in Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS), Washington, DC, USA, 2005, IEEE Computer Society.
[73] K.-U. Sattler, P. Rösch, E. Buchmann, and K. Böhm, *A Physical Query Algebra for DHT-based P2P Systems*, in Proceedings of the 6th Workshop on Distributed Data and Structures, Lausanne, Switzerland, July 2004.
[74] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, *Coda: A Highly Available File System for a Distributed Workstation Environment*, IEEE Transactions on Computers, 39 (1990), pp. 447–459.
[75] T. Seidmann, *Replicated Distributed Shared Memory For The .NET Framework*, in Proceedings of 1st Int.Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing, Plzen, Czech Republic, February 2003.
[76] Stefan Saroiu, Krishna P Gummadi, Richard J Dunn, Steven D Gribble, and Henry M. Levy, *An Analysis of Internet Content Delivery Systems*, SIGOPS Operating Systems Review, 36 (2002), pp. 315–327.
[77] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, *A Survey of Peer-to-Peer Content Distribution Technologies*, ACM Computing Surveys, 36 (2004), pp. 335–371.
[78] H. Stockinger, *Distributed Database Management Systems and the Data Grid*, in MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies, Washington, DC, USA, 2001, IEEE Computer Society, p. 1.
[79] H. Stockinger, A. Samar, K. Holtman, W. E. Allcock, I. Foster, and B. Tierney, *File and Object Replication in Data Grids.*, Cluster Computing, 5 (2002), pp. 305–314.
[80] Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Junwen Lai, Yilei Shao, Chi Zhang, Elisha Ziskind, Arvind Krishnamurthy, and Randolph Y. Wang, *Segank: A Distributed Mobile Storage System*, in FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2004, USENIX Association, pp. 239–252.
[81] Sun Microsystems, *JS—JavaSpaces Service Specification*. `http://java.sun.com/products/jini/2.0/doc/specs/html/js-spec.html` 2001.
[82] Sushant Goel, Hema Sharda, and David Taniar, *Atomic Commitment and Resilience in Grid Database Systems*, International Journal of Grid and Utility Computing, 1 (2005), pp. 46–60.
[83] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Ka diyska, G. Miklau, and P. Mork, *The Piazza Peer Data Management Project*, SIGMOD Record, 32 (2003).
[84] D. B. Terry, K. Petersen, M. Spreitzer, and M. Theimer, *The Case for Non-transparent Replication: Examples from Bayou.*, IEEE Data Engineering Bulletin, 21 (1998), pp. 12–20.
[85] Tevfik Kosar and Miron Livny, *Stork: Making Data Placement a First Class Citizen in the Grid*, in ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), Washington, DC, USA, 2004, IEEE Computer Society, pp. 342–349.
[86] P. Valduriez and E. Pacitti, *Data Management in Large-Scale P2P Systems.*, in VECPAR, M. J. Daydé, J. Dongarra, V. Hernández, and J. M. L. M. Palma, eds., vol. 3402 of Lecture Notes in Computer Science, Springer, 2004, pp. 104–118.
[87] S. Venugopal, R. Buyya, and K. Ramamohanarao, *A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing*, ACM Computing Surveys, (2006). To appear.
[88] Venugopalan Ramasubramanian and Emin G Sirer, *Exploiting Power Law Query Distributions for O(1) Lookup Performance in Peer to Peer Overlays*, in Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI), USENIX Association, 2004.
[89] ———, *The Design and Implementation of a Next Generation Name Service for the Internet*, in SIGCOMM '04: Proceedings

of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2004, ACM Press, pp. 331–342.
[90] Weijian Fang, Cho-Li Wang, and Francis C M Lau, *On the Design of Global Object Space for Efficient Multi-threading Java Computing on Clusters*, Parallel Computing, 29 (2003), pp. 1563–1587.
[91] Weimin Yu and Alan Cox, *Java/DSM: A Platform for Heterogeneous Computing*, in ACM 1997 Workshop on Java for Science and Engineering Computation, June 1997.
[92] Xueyan Tang and Jianliang Xu, *QoS-Aware Replica Placement for Content Distribution*, IEEE Transactions on Parallel and Distributed Systems, 16 (2005), pp. 921–932.
[93] B. Yang and H. Garcia-Molina, *Designing a super-peer network.*, in Dayal et al. [28], pp. 49–62.
[94] Yi Lin, Bettina Kemme, Marta Patino-Martinez, and Ricardo Jimenez-Peris, *Middleware Based Data Replication Providing Snapshot Isolation*, in SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, New York, NY, USA, 2005, ACM Press, pp. 419–430.
[95] L. Zhenyun Zhuang and Member-Yunhao Liu, *Dynamic Layer Management in Superpeer Architectures*, IEEE Transactions Parallel and Distributed Systems, 16 (2005), pp. 1078–1091.

Edited by: Thomas Ludwig
Received: May 25, 2006
Accepted: October 11, 2006