JOURNAL OF OBJECT TECHNOLOGY
Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2002

Vol. 5, No. 7, September–October 2006

Scaling a Shared Object Space to the Internet: Case Study of Virat

A Vijay Srinivas and **D Janakiram**
{avs@cs.iitm.ernet.in}, {djram@cs.iitm.ernet.in}
Distributed & Object Systems Lab,
Department of Computer Science & Engineering,
Indian Institute of Technology, Madras, India
http://dos.iitm.ac.in

Scalability is an important issue in the construction of distributed systems. Shared object spaces provide an elegant and easy-to program abstraction for building applications. However, existing shared object spaces have been realized at the cluster level. Use of centralized components, lack of effective failure handling mechanisms, lack of efficient object lookup mechanisms as well as consistency maintenance are the key issues that inhibit scalability of existing shared object spaces. We present the case study of scaling an existing shared object space (Virat) to the Internet. Bottlenecks in Virat include the granularity of consistency maintenance and Object Meta-data Repository (OMR) failures. Both the design and implementation of Virat has been modified in order to increase the granularity at which consistency is maintained. Virat has also been redesigned such that the OMRs form a Peer-to-Peer (P2P) overlay in order to handle OMR failures and improve scalability. Experimental evaluations are presented to show that the optimized version of Virat scales better, especially over a wide-area network. In addition, this paper also explains how to develop applications over the shared object space, with code sketches.

1 INTRODUCTION

Distributed Shared Memory (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 [1]. Collaborative applications such as on-line chatting and collaborative browsing would be easier to develop over a DSM.

Java/DSM [2] provides a Java Virtual Machine (JVM) abstraction over TreadMarks. It is an example of page based DSMs, similar to Munin [3] and TreadMarks. Page based DSMs shared data at the level of memory pages, while object based DSMs (also known as shared object spaces) share application objects. 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. Object based DSMs alleviate the problem by a more

Cite this document as follows: A Vijay Srinivas and D Janakiram: *Scaling a Shared Object Space to the Internet: Case Study of Virat*, in Journal of Object Technology, vol. 5, no. 7, September–October 2006, pages 75–95,
http://www.jot.fm/issues/issues_2006_08/article4

application specific granularity of sharing. Examples of object based DSMs include Linda [4] and Orca [5]. To our best knowledge, none of the shared object spaces have been really scaled to the Internet, as we argue below. An Internet scale shared object space can make it easier to develop applications such as computer supported cooperative work (CSCW) based collaborative designing and multi-player games [6], among others.

Global shared objects [7] 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 a per-object lock manager for keeping replicas consistent. It does not address failures of the lock manager. Java spaces specification from Sun [8] provides a distributed persistent 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.

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 [9][1], the study was done on a Myrinet cluster. This overhead may become substantial and affect application performance adversely over the Internet. This is primarily because the totally ordered group communication relies on a sequencer or uses broadcast. These could be expensive over the Internet. But Orca has not been evaluated on the Internet scale.

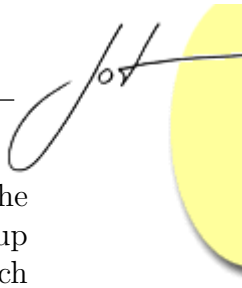
We have proposed a generic scalability model for analyzing distributed systems [10]. It takes the view that scalability of distributed systems should be analyzed considering related issues such as consistency, synchronization, and availability. By applying the scalability model on shared object spaces, we have identified the key bottlenecks that inhibit existing shared object spaces from scaling up to the Internet:

- Centralized Components
  Many existing DSMs and shared object spaces have some centralized components that affects their scalability. For instance, Orca has a sequencer for realizing totally ordered group communication, while others like T Spaces [11] 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

[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.



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 uses centralized lookup mechanisms. Object lookup mechanisms in distributed object middleware such as CORBA, 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 mechanism such as release consistency and entry consistency. Relaxed consistency mechanisms have also been explored in other areas [12, 13]. 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 [14] and Tapestry [15] assume replicas are read-only.

Scaling up a Shared Object Space to the Internet

We have developed Virat [16], that was intended as a wide area shared object space to addresses infrastructure support for large scale distributed applications. It supports various object interaction styles so that applications can be architected easily. Virat uses a novel mechanism for handling failures and provides a data centric Concurrency Control (CC) mechanism to realize various consistency models. Virat has been extended to a fully typed shared event space, facilitating publish-subscribe kind of interactions and large scale event notifications [17]. However, by applying the scalability model [10] to Virat, we find that there are bottlenecks. The goal of this paper is to explain how we address the above mentioned problems in the context of Virat and scale it up to the Internet.

Peer-to-Peer systems such as Pastry [14] have been proposed to handle efficient, decentralized object lookups and tolerate failures. The early versions of Virat handled OMR failures through explicit lookup servers, under the assumption that the lookup server and the OMR in that cluster do not fail simultaneously. We have redesigned Virat to handle OMR failures elegantly and scale up better. 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. Information maintained in any OMR is also stored in k-replicas. The former enables the lookup time to be only O(log(n)) for n nodes and the latter enables Virat to handle OMR failures. Virat has been integrated with the code of Freepastry (available at http://freepastry.rice.edu/) for object lookup and routing among the OMRs. Performance evaluation over a wide-area network shows that the P2P communication based Virat scales better.

We also find that Virat could be optimized with respect to consistency granularity maintenance. We evaluate its performance and show that the optimized Virat scales better. In the process, we experimentally compare three different design mechanisms for achieving consistency in Virat: a two phase commit (2PC) based

naive mechanism, a CompareAndSwap (CAS)-like update mechanism and the optimized mechanism. We find that the CAS-like mechanism, although it performs well, degenerates to the optimized mechanism to handle failures and scale up.

Section 2 provides a brief overview of the existing Virat shared object space. Section 3 describes the scalability model that we have applied to Virat. Section 4 explains the changes in design and implementation made to Virat for scaling it up. Section 5 presents the performance evaluation of Virat over an Institute wide as well as a wide-area testbed. Section 6 compares the modified Virat with a few other large scale systems. Section 7 puts the paper in perspective and provides directions for future research.

2 VIRAT: A SHARED OBJECT, EVENT AND SERVICE SPACE

Virat supports middleware services such as naming and trading as well as replica object management. Virat uses an independent check pointing and lazy reconstruction mechanism to handle failures of OMRs. Lookup servers are also run, one in each cluster. The lookup server maintains the current location of the OMR in that cluster. Under the assumption that both lookup servers and OMRs in a cluster would not fail simultaneously, failures are handled. The OMRs (one per cluster) are responsible for cluster level management of replicas. The OMRs communicate among themselves to locate objects across clusters.

The appendix explains how to write programs over the shared object space. 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 OMR services and returns a replica of the object to the client. The OMR service maintains information about objects and the current accessors for each object. Each object has a data counter, for ensuring data centric CC [18]. The data counter is also used as a versioning mechanism that forms the basis of reconstruction of global system states, while using independent checkpointing to recover from failures.

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 of this cluster. 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.

3 THE SCALABILITY MODEL

A number of studies have been made considering the scalability of distributed systems. A scalability metric based on productivity was defined in [19]. The metric evaluates scalability as the product of throughput and response time (or any value function) divided by the cost factor. An analytical model of server systems based on Layered Queuing Networks (LQN) was proposed in [20]. A more general analytical model of client-server systems based on rendezvous networks was given in [21]. It illustrates how certain slow servers could become bottlenecks and suggests threading/cloning to relieve these bottlenecks. Recent work on design patterns for concurrent and networked systems documents patterns such as *Reactor, Half-Sync Half-Async* for efficient multi-threading in server systems [22]. Scalability of Java 2 Enterprise Edition (J2EE) technology has been evaluated in [23].

All the above efforts perceive scalability in isolation and some are even specific to certain technologies. The inter-related issues of synchronization, consistency and availability are neglected. One of our main conjectures is that availability, consistency, synchronization and scalability are closely related and should be looked at in totality, not in isolation. We have proposed an analytical model that considers scalability as a function of synchronization, consistency, availability, workload and faultload [10].

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

- *avail* is availability - can be quantified as the ratio of number of transactions accepted versus submitted. Availability itself is a function of network availability (number of operations reaching any replica) and service availability (number of operations accepted by a replica). If the availability requirements are very high, a trade-off between scalability and consistency may be required. If strict consistency is also required, the scalability of the system may be compromised. If consistency can be relaxed, both high availability and scalability can be achieved [24].

- *consis* is consistency, itself a function of update ordering and consistency granularity. Update ordering refers to update ordering mechanisms across replicas and can be one of causal, serializable or PRAM. Consistency granularity refers to the grain size at which consistency needs to be maintained in the system. There are two dimensions to consistency granularity. Caching is an important and well known method for improving performance. The size of caching is an important factor governing the performance. A higher size implies more chances of program locality effects and better performance. The other dimension is the granularity of consistency maintenance; whether it is at the individual replica level or using some form of aggregation. It may be easier to maintain consistency at aggregate level than at individual replica level, but at the cost of strictness.

- *sync* refers to synchronization among the replicas. The two dimensions of synchronization are how often the replicas are synchronized and the mode of synchronization. The former is captured in terms of the δ value [25], δ referring to the number of updates that can be buffered in a particular node before synchronizing with others. The latter can be modeled as a γ value, with $\gamma = 0$ implying a server push/invalidation model and a positive non-zero value indicating that a replica could miss some updates and can be updated lazily, either in push or pull mode. This may reduce the messaging overhead and improve the scalability of the system. A similar idea has been exploited in the notion of *local consistency* [13], but without any attempt at quantification.

- Workload can be broken down into workload intensity and workload service demand characterization. Workload intensity refers to number of transactions per second or number of clients etc. and service demand refers to CPU time for operations, network delays etc. As workload increases, performance of the system must degrade gracefully. Improving service demand (say by having a faster processor) can help in graceful degradation.

- Faultload is the failure sequences and number as well as location of replicas. In Internet scale systems, failures become important. Systems as well as networks are prone to failure, with increased message latencies creating additional difficulties.

The function f could be a mathematical function that can be used to quantify the scalability of a distributed system. At a higher level, the f can also be viewed as an algorithm. The primary goal of this algorithm is to identify scalability bottlenecks in the system. An open issue is whether the arguments to f are complete and minimal.

4 REDESIGNING VIRAT TO THE INTERNET SCALE

The scalability model is a useful tool to identify bottlenecks in distributed systems. The scalability model has been applied to Virat and it gives directions for optimization of Virat. First, relating to the consistency parameter, Virat has been optimized to implement different update ordering mechanisms. These imply different consistency mechanisms, namely causal consistency, sequential consistency and causal serializability. This is in line with the established principle that relaxing consistency ordering leads to improved scalability [13]. The stricter the consistency mechanism that needs to be enforced, the more overhead is involved in terms of messages and protocols. For instance, to enforce serial consistency, a two phase commit (2PC) or a more sophisticated three phase commit (3PC) protocol may have to be in place. Causal consistency can be implemented by a data centric mechanism [26]. Causal serializability can also be implemented without a system wide agreement protocol.

A further optimization in Virat relates to consistency granularity. Serial consistency is enforced by a 2PC protocol. The DSM objects in each node which maintain

replicas of an object, run the 2PC among themselves. The initiator node acts as a 2PC coordinator. 2PC may be expensive, especially in a wide-area setting. An optimization is to ensure strict consistency only at the level of OMRs. They are responsible for update propagation to other nodes. Thus, a serializability protocol, if required is enforced only among the repositories and not among all replicas. This reduces the number of participants in the 2PC algorithm, making it more efficient. The OMRs are responsible for propagating updates to individual replicas. An invalidation based approach is used to maintain consistency among the OMRs, to avoid 2PC. These optimizations are expected to improve the scalability of Virat.

Fault tolerance is the other dimension along which Virat has been redesigned. 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 predesignated node as an OMR. To handle OMR failure, a lookup server was used. The lookup server keeps update 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 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.

Design and Implementation

This section presents the details of redesigning Virat along the lines of consistency granularity and fault-tolerance (P2P routing among OMRs) for scalability. We now explain the design and implementation details of the three consistency mechanisms: naive 2PC mechanism, a CAS-like update mechanism and an optimized 2PC mechanism. The client instantiates a DSM object and makes calls on that object. All the calls are forwarded to a local DSM runtime object, which in turn forwards them to the OMR whenever necessary. The DSM runtime object contains data structures to

maintain the objects created or read by the local node. The OMR maintains details of objects accessed within a cluster.

Interface Additions for Naive 2PC Based Consistency Mechanism

The methods to be added to the interface of the DSM object are given below, with brief explanations of their purpose.

- String WriteSequential (Object obj, String ClassName, Object[] dependencyList)

This method allows an application to write into an object and ensure sequential consistency. The normal *Write* method of Virat ensures only causal consistency by default. This method makes the DSM object act as a 2PC coordinator and initiate a 2PC algorithm to ensure that the write is propagated to all replicas consistently.

- public void UpdateClientList(String oid, server.returnable clientList)

This method is called from a *Create* method (this creates a new replica). It adds the new replica to all the existing clients, so that whichever client initiates a 2PC, this client is also included as a participant.

- public boolean Prepare2PC(String oid, Object obj)

This method refers to the prepare phase of 2PC among replicas to serialize writes. This method is invoked by the 2PC coordinator. It checks local constraints and returns true if there are no other updates pending. It returns false otherwise, in which case the 2PC coordinator will abort in the next phase.

- public void Commit2PC(String oid, Object obj)

This method refers to the commit phase of 2PC among replicas to serialize writes. This method is invoked by the 2PC coordinator, after it gets a 'true' from all replicas. Even if it gets a single 'no' vote, the 2PC has to be aborted.

- public void Abort2PC(String oid, Object obj)

This is similar to the above for the abort phase of 2PC.

CompareAndSwap(CAS)-like Updates

The naive 2PC based consistency mechanism of Virat can be compared to the Distributed Data Structure (DDS) approach of [27]. DDS was proposed as a centralized interface to persistent distributed data for web services. DDS is actually based on replication and distribution and provides atomic single element updates. The consistency of the replicas are maintained by a 2PC protocol to ensure serialization. This makes write operations very expensive and a bottleneck for scalability. Thus, an efficient mechanism for updating replicas would be very useful.

We have developed a CAS-like primitive that is inspired from multi-processor synchronization primitives [28]. We have implemented the CAS-like update mechanism by using a hierarchical tree-based multilevel CAS. It provides only serializability semantics. The main idea is that CAS-like update mechanism is optimistic, compared to the pessimistic approach of 2PC or mutual exclusion. A simple way to realize the CAS-like update mechanism is to have only a single node (owner) at the top of the hierarchy. A CAS operation is done at each level of the hierarchy and propagated up, if the local CAS succeeds. If the local CAS fails, the CAS returns failure. Different nodes may be owners for different data items for load distribution. However, this mechanism may not be able to handle failures. To handle failures, multiple nodes may be required at the root of the hierarchy. But this may lead to consistency problems at the root set. The interfaces required to realize this CAS-like update mechanism is given below.

- Object CompareAndSwap(String oid, Object oldValue, Object newValue)

This is the CAS method on the OMR. This method is invoked by a CAS operation from the child. If local CAS succeeds, it sends a CAS request to the owner of the object with id as oid. If local CAS fails, it returns a failure. There is a similar CAS method on the DSM runtime object, which is invoked by the application. The DSM runtime object invokes CAS on the OMR of the cluster.

Optimized Consistency Mechanism - OMR Level 2PC

Consistency is maintained at the level of OMRs only, and they are responsible for update propagation to other nodes. Thus, serializability protocol, if required is enforced only among the repositories and not among all replicas. The number of nodes among which agreement must be reached for serializing writes is decreased drastically. As a result, it is much easier to implement. Further, as the number of nodes is higher, greater is the failure probability. 2PC will be even more expensive given failure scenarios. The optimized mechanism is hence, expected to scale better, especially in a wide-area setting (in spite of its dynamic nature). However, this method does not ensure strict serializability among all replicas. It is a different way of relaxing consistency compared to the normal ways [12]. Applications which can

tolerate slight inconsistencies such as those given in [29] can benefit. The interface changes required to realize this mechanism are:

- The methods Prepare2PC, Commit2PC and Abort2PC on the DSM runtime object must be moved to the OMR.
- public boolean WriteObjRepoSeq(String oid,Object obj) method must be added to the OMR. This method will initiate the 2PC among the OMRs, with this OMR acting as 2PC coordinator. The *WriteSequential* method on the DSM object must be modified to invoke this method of the OMR.
- void notifyLocalSubscribers(Object obj, String eventType) method of the OMR[2] must be modified to propagate updates to objects. Basically, the event space of Virat [17] is being used for update propagation by the OMRs within clusters.

Incorporating Pastry Routing Among OMRs

The Freepastry implementation of Pastry provides the *commonapi* class. This class is an abstraction of common APIs for P2P systems. Important methods of this class include *route(Id key, Message msg, NodeHandle hint) and replicaSet(Id id, int maxRank)*. The *route* method implements the traditional P2P routing, ensuring that the message is routed to the node with the closest matching id that is alive. The *deliver* message on the receiving *Application* instance is invoked. The *replicaSet* returns a set of nodes on which replica meta-data of a given object can be stored.

If an OMR invokes the *route* method, the *deliver* message on the application process at the destination OMR is invoked. However, the originating OMR needs a handle to the destination OMR to invoke methods such as *Read*, *Write* etc. that are part of the OMR interface. Since, the route method does not return a handle, in our implementation the *deliver* method invokes the *passMyHandle* on the originating OMR to pass its own reference. The originating OMR makes invocations such as *GetObjectDetailsWithID* (to get details of an object created under a different OMR) on the handle.

Each OMR maintains a list of OMRs (since it gets an invocation from the OMR, it can get the IP address) for all objects for which it stores meta-data. This list is used for updating OMRs for *Write* operations. Currently, we provide only causal consistency in the P2P based version of Virat. It should be noted that for implementing causal consistency, local operations are sufficient and global agreement protocols may not be necessary [30]. Alternative consistency mechanisms for P2P systems have not been explored too much in the literature, with many current P2P systems such as Tapestry [15] assuming data is read-only. We leave it for future research to realize other consistency models in the P2P version of Virat.

[2]This method normally notifies events to the local subscribers (who are within this cluster).

An optimization to avoid 2PC among OMRs has been realized. When an OMR needs to write an object, instead of becoming a 2PC coordinator and realizing 2PC for serializing writes, the OMR sends an invalidate message to other OMRs. When OMRs receive invalidate message, they invalidate the local copy of the object. A pull based model is used for updating the values, implying that other OMRs can pull the values from the current writer. The OMRs are responsible for propagating the updates to the replicas within their cluster.

5 PERFORMANCE STUDIES

This section details some of the performance studies we have conducted over Virat to show its scalability. The initial set of experiments have been conducted on an Institute wide network, consisting of about thirty 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 Ethernet connection. A few nodes from an engineering college in Trichy[3] were used and a wide-area testbed was formed. This testbed was used for the wide-area results reported in this paper.

Figure 1 shows the overhead of the *WriteSequential* method call with increasing number of nodes and hence clients (each node can be hosting hundreds of clients). It can be observed that the naive Virat incurs nearly linear overheads. This non-trivial overhead is more prominently seen in the wide area results. Thus, as the number of Internet nodes participating in Virat increases, the naive Virat may have trouble scaling up. This is due to the overhead involved in the 2PC algorithm that is required to serialize the writes.

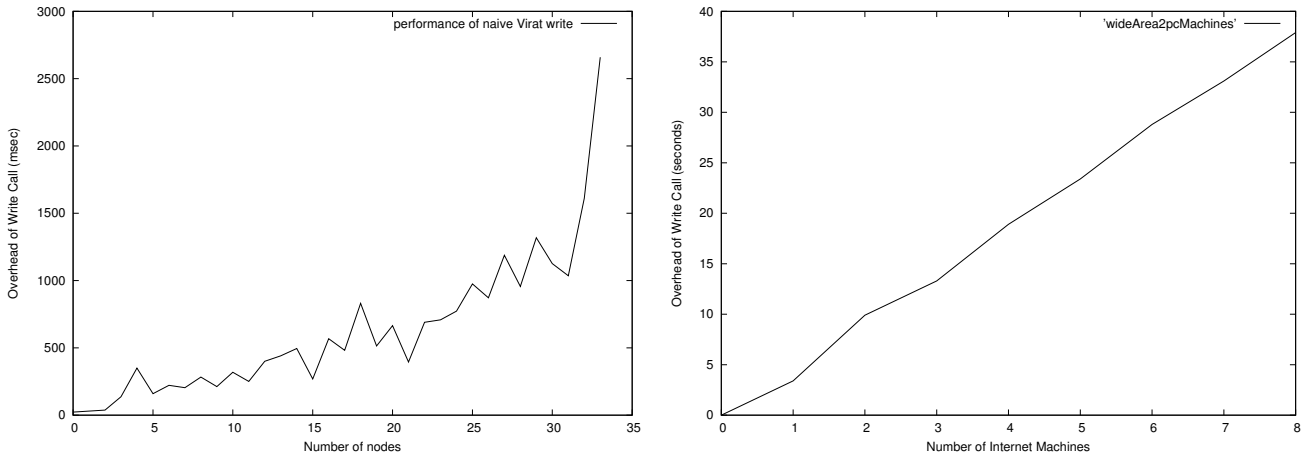


Figure 1: Write Overheads in Naive Virat on Campus Intranet and Wide-Area Testbeds

[3]Another city in Tamil Nadu, about 200 miles from Chennai.

Figure 2 shows the overhead of the *WriteSequential* method call for the CAS-like update mechanism, with only one node at the top of the tree. It can be observed that this method performs far better, that too with increasing number of nodes. However, the fact that there is only one node at the top of the hierarchy implies that this mechanism is prone to failures, both node as well as network failures. If the number of nodes forming the top of the hierarchy is increased, failures can be tolerated better. But, this leads to issues of synchronization between the top level nodes. This update mechanism becomes similar to the optimized Virat mechanism (with a 2PC at object repository level), which handles consistency such issues.

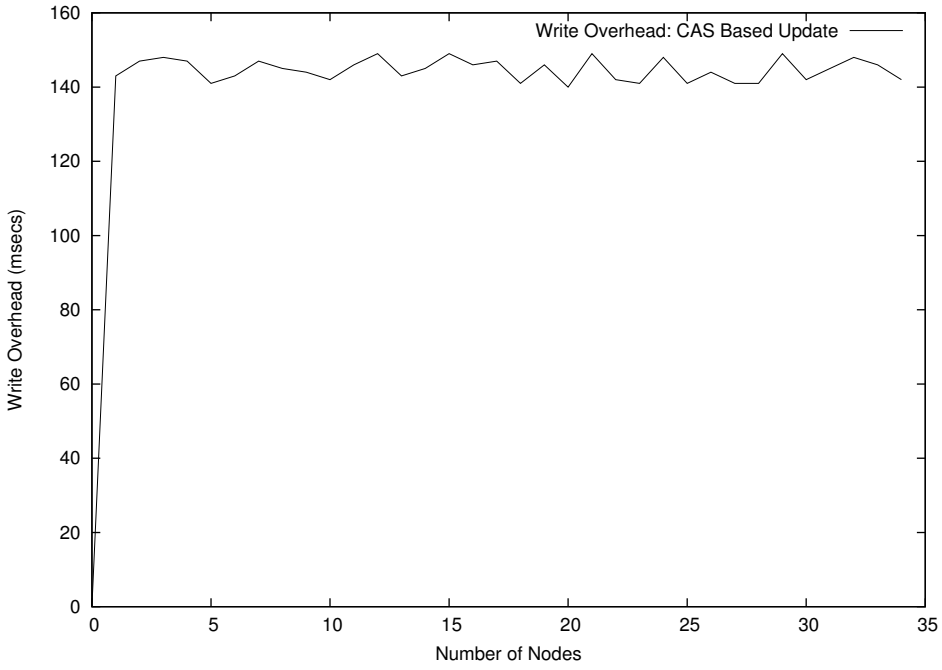


Figure 2: Performance of CAS-like Update Mechanism on Campus Intranet

Figure 3 shows the performance of the *WriteSequential* method in optimized Virat. Since the overheads of 2PC are reduced by increasing the granularity of consistency, the performance degrades slowly, even in the wide-area testbed.

The overhead of object replication has been measured in both the naive Virat (Virat without P2P routing between OMRs) and the P2P based Virat. These overheads have been measured by creating an object replica (by the *Create* method call on the Virat interface) from an OMR in a different cluster than the one on which the object was originally created. The originating OMR sequentially searches through each of the other OMRs to find which OMRs maintain information about this object in naive Virat. In the P2P version, the originating OMR does a P2P routing to find the OMR which maintains meta-data about this object. Consequently, the overhead of object replication across clusters is much higher in naive Virat compared to the P2P based Virat. This is depicted in figure 4.

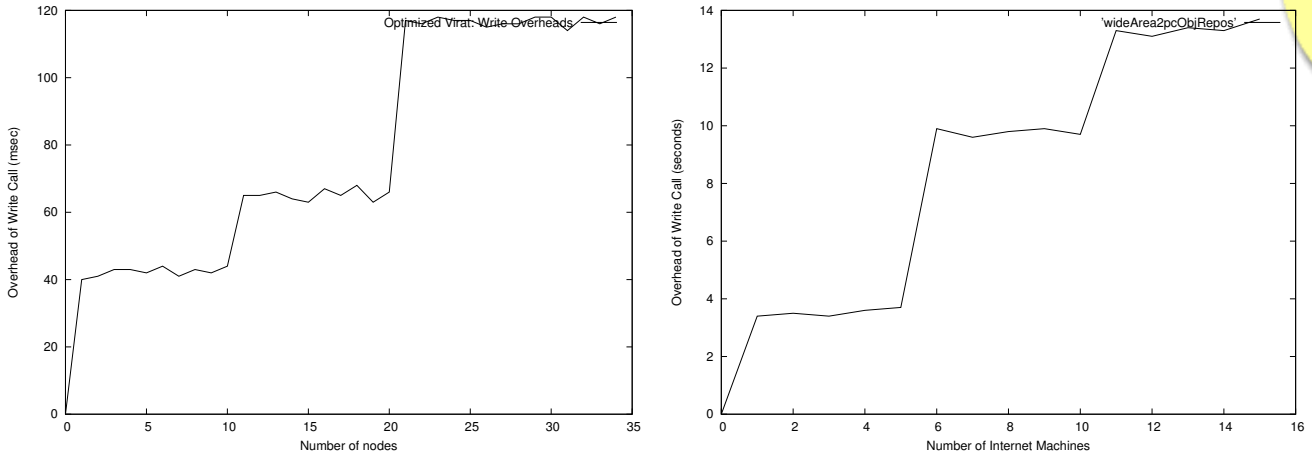


Figure 3: Performance of Optimized Virat on Campus Intranet and Wide-Area Testbeds

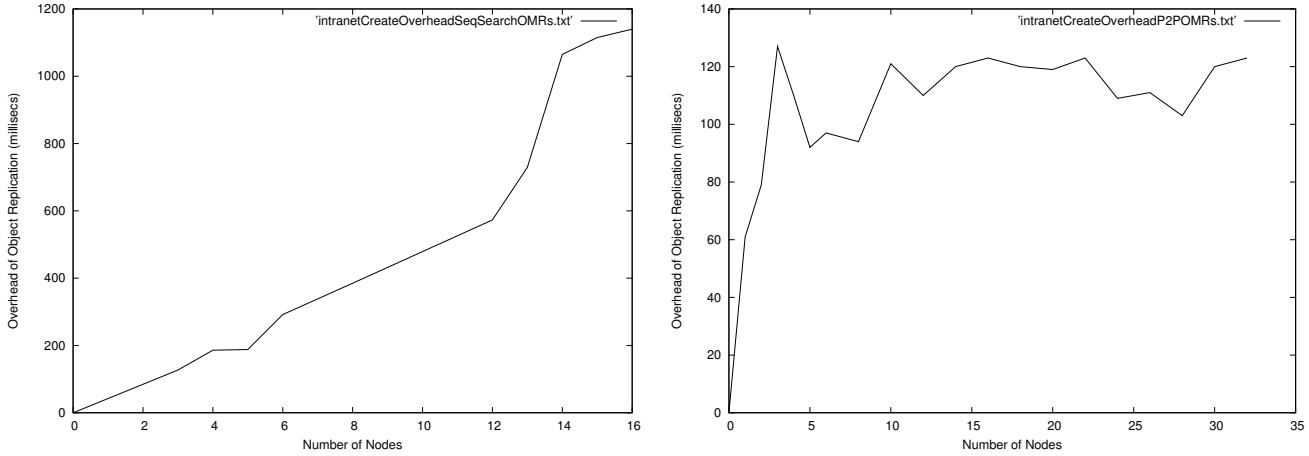


Figure 4: Object Replication Overhead in Naive Virat and P2P Based Virat on Campus Intranet Testbed

6 RELATED WORK

Two other models have endeavoured to extend DSM across clusters: JDSM [31] and Lemuria [32]. JDSM presents an implementation of software DSM written in Java that can be ported to several cluster platforms. JDSM attempts to provide complete distribution transparency, which means that the programmer is totally transparent to the DSM. This results in non-trivial overheads to check object accesses. This, coupled with cache coherency maintenance costs and lack of adequate failure handling mechanisms limits the scalability of JDSM. Lemuria uses a new consistency criteria called *cluster based release consistency* to enable the DSM to be used in a wide-area environment. However, no detailed study of the performance and scalability of these DSMs have been documented in the literature.

T spaces is a shared object space from IBM [11] that adds database functionality

to Linda tuplespace [4] 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*. However, scalability is not one of the design goals of T spaces nor has it been evaluated for scalability. It may have difficulty scaling up, as it uses *tspace server* as a centralized component. Even if an application uses multiple tspace servers, the communication between these servers is not P2P. Hence, just like a naive Virat where communication between OMRs was not P2P, T spaces may not scale. Further, it does not address failures. At the Internet scale, failures become a serious concern.

Legion is a middleware approach targeted at providing a global virtual computer abstraction, that is built on top of Mentat, a parallel programming platform. Legion addresses wide area scalability issues explicitly. However, it does not provide distribution transparency to applications. It is well known that though object request brokers such as CORBA provide distribution transparency and various services such as transactions, replication and fault tolerance is not easily handled. FT-CORBA, the Fault Tolerance specification of CORBA [33] which addresses replication, still has serious limitations with respect to: nondeterminism, transparency, three kinds of state (application, POA, infrastructure), weak identity, replication of clients and factories etc. [34].

Globe [35] is a middleware for constructing wide-area applications that addresses scalability issues explicitly. However, for scaling up to the wide-area, it needs scalable object location services as documented in [35]. They have addressed scalable location services for Globe in [36]. The location service is based on a worldwide distributed search tree that can be compared to the Distributed Hash Table (DHT) on which many structured P2P systems [37] including Pastry and consequently Virat is based on. The tree based approach is scalable in terms of performance but does not handle node failures, whereas DHTs handle both.

7 CONCLUSIONS

We have analyzed both the geographic and numerical scalability of Virat, a shared object space by using an analytical model that we had proposed earlier. We had to redesign (and re-implement) Virat along two dimensions: consistency granularity - maintain consistency at the level of OMRs; fault tolerance - make the OMRs form a P2P overlay to handle failures and scale up better. Thus, Virat becomes a unique scalable P2P based wide-area shared object space. To the best of our knowledge, shared object space or Distributed Shared Memory (DSM) implementations have not been realized over P2P systems, nor scaled to the wide-area. We have also presented performance studies over a wide-area testbed to show the scalability improvements made in Virat.

An interesting future direction is to make Virat a completely P2P shared object space, instead of making only the OMRs form a P2P overlay. All the nodes could

form the overlay, with OMRs being super-peers. We are integrating Virat into Vishva, a two layered P2P routing substrate that we have developed as a middleware for grid computing systems [38]. This would enable OMR data to be replicated within a *zone* and make it easier to keep the data consistent. We are also building a replication service for large data grids over Virat and Vishva. This is a step towards our end goal of realizing large data grids over Virat. The idea of building grids over shared object spaces is also being exploited by other researchers [39].

REFERENCES

[1] C. Amza, A.L. 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(2):18–28, February 1996.

[2] 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.

[3] John B Carter. Design of the Munin Distributed Shared Memory System. *Journal of Parallel and Distributed Computing*, 29(2):219–227, February 1995.

[4] Nicholas Carriero and David Gelenter. Linda in Context. *Communications of the ACM*, 4(32):444–458, April 1989.

[5] 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(3):190–205, 1992.

[6] Tsun-Yu Hsiao and Shyan-Ming Yuan. Practical Middleware for Massively Multiplayer Online Games. *IEEE Internet Computing*, 9(5):47–54, September/October 2005.

[7] 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(11-12):1563–1587, 2003.

[8] Sun Microsystems. JS - JavaSpaces Service Specification. http://java.sun.com/products/jini/2.0/doc/specs/html/js-spec.html, 2001.

[9] 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(1):1–40, 1998.

[10] A Vijay Srinivas and D Janakiram. A Model for Characterizing the Scalability of Distributed Systems. *ACM SIGOPS Operating Systems Review*, 39(3):64–72, 2005.

[11] P Wyckoff, S W McLaughry, T J Lehman, and D A Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[12] 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.

[13] Mustaque Ahamad and Rammohan Kordale. Scalable Consistency Protocols for Distributed Services. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):888–903, September 1999.

[14] Antony Rowstron and Peter 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)*, pages 329–350, Heidelberg, Germany, November 2001.

[15] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.

[16] A Vijay Srinivas, D Janakiram, and Raghevendra Koti. Virat: An Internet Scale Distributed Shared Object, Event and Service Space. Technical Report IITM-CSE-DOS-2004-03, Distributed & Object Systems Lab, Indian Institute of Technology, Madras, 2004.

[17] A Vijay Srinivas, Raghavendra Koti, A Uday Kumar, and D Janakiram. Realizing Large Scale Distributed Event Style Interactions. In *Proceedings of the European Conference on Object Oriented Programming (ECOOP) Workshop on Communication Abstractions for Distributed Systems*. Oslo, Norway, 2004.

[18] D Janaki Ram, N S K Chandra Sekhar, and M Uma Mahesh. A Data-Centric Concurrency Control Mechanism for Three Tier Systems . In *Proceedings of the IEEE Symposium on Web caching for E-business, part of IEEE Conference on Systems, Man and Cybernetics*. Arizona, USA, 2001.

[19] Prasad Jogalekar and Murray Woodside. Evaluating the Scalability of Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, June 2000.

[20] J A Rolia and K C Sevcik. The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, August 1995.

[21] J E Neilson, C M Woodside, D C Petriu, and S Majumdar. Software Bottlenecking in Client-Server Systems and Rendezvous Networks. *IEEE Transactions on Software Engineering*, 21(9):776–782, September 1995.

[22] Douglas Schmidt, Michael Stal, Hans Robert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., 2000.

[23] Paul Brebner and Jeffrey Gosper. How scalable is J2EE technology? *ACM SIGSOFT Software Engineering Notes*, 28(3):4–10, May 2003.

[24] Eric Brewer. Lessons from Giant Scale Services. *IEEE Internet Computing*, 5(4):46–55, July/August 2001.

[25] Chi Zhang and Zheng Zhang. Trading Replication Consistency for Performance and Availability: an Adaptive Approach. In *Proceedings of the 23rd International Conference on Distribut ed Computing Systems*. Providence, Rhode Island, USA, May 2003.

[26] D. Janaki Ram, M. Uma Mahesh, N.S.K. Chandra Sekhar, and Chitra Babu. Causal Consistency In Mobile Environment. *ACM Operating Systems Review*, 35(1):34–40, January 2001.

[27] Steven D Gribble, Eric A Brewer, Joseph M Hellerstein, and David Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *4th Symposium on Operating System Design & Implementation*. San Diego, California, USA, October 2000.

[28] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-Free Synchronization: Double-Ended Queues as an Example. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003)*. Providence, Rhode Island, USA, May 2003.

[29] Mustaque Ahamad, Ranjit John, Prince Kohli, and Gil Neiger. Causal memory meets the consistency and performance needs of distributed applications! In *EW 6: Proceedings of the 6th workshop on ACM SIGOPS European workshop*, pages 45–50, New York, NY, USA, 1994. ACM Press.

[30] Mustaque Ahamad, Phillip W Hutto, Gil Neiger, James E Burns, and Prince Kohli. Causal memory: Definitions, implementation and programming. Technical Report GIT-CC-93/55, Georgia Institute of Technology, 1994.

[31] Yukihiko Sohda, Hidemoto Nakada, Hirotaka Ogawa, and Satoshi Matsuoka. Implementation of Portable Software DSM in Java . In *Joint ACM Java Grande - ISCOPE 2001 Conference*, pages 163–172. Palo Alto, CA , USA, June 2001.

[32] Saito Shoichi and Kunieda Yoshitoshiand Ojubo Eiji. An Implementation of Distributed Shared Memory in Wide Area Network and Its Performance Evaluation. *IPSJ JOURNAL*, 6(40), 2001.

[33] Object Management Group. Fault Tolerant CORBA (Final Adopted Specification), December 2001. formal/01-12-29.

[34] Pascal Felber and Priya Narasimhan. Experiences, Strategies, and Challenges in Building Fault-Tolerant CORBA Systems. *IEEE Transaction on Computers*, 53(5):497–511, May 2004.

[35] Maarten van Steen and Philip Homburg and Andrew S. Tanenbaum. Globe: A Wide-Area Distributed System . *IEEE Concurrency*, 7(1):70–78, January-March 1999.

[36] M Van Steen, F J Hauck, P Homburg, and A S Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, 36(1):104–109, January 1998.

[37] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[38] M Venkateswara Reddy, M A Maluk Mohamed, Tarun Gopinath, A Vijay Srinivas, and D. Janakiram. Vishwa: A Paradigm for Anonymous Remote Computation and Communication for Peer-to-Peer Grid Computing. Technical Report IITM-CSE-DOS-2005-12, Distributed & Object Systems Lab, Department of Computer Science & Engineering, Indian Institute of Technology Madras, 2005.

[39] Tobin J Lehman and James H Kaufman. Optimal Grid: Middleware for Automatic Deployment of Distributed FEM Problems on an Internet-Based Computing Grid. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03)*, pages 164– 171, Kowloon, Hong Kong, December 2003.

APPENDIX: PROGRAMMING WITH VIRAT SHARED OBJECT SPACE

We first illustrate how the client program will look like. This program creates an object and makes it sharable across nodes. The code to be written for accessing this shared object from a different node is shown next. The application is a simple chat application. It is taken for illustrative purposes and one can develop complex applications over Virat.

```
class createSharedObject
{
  public static void main(String args[])
  {

   try {
      dclass chatObject = new dclass();
      Dsm dsm = new Dsm();
      String oid = dsm.CreateNewObject(chatObject ,"dclass");
   // this means we want to share the dc object.
   ..... // other code.
   // accept user input for next line
      while (!done)
      {
         try
            {
               ch = System.in.read();
               if (ch < 0 || (char)ch == '\n')
                  done = true;
               else
                  userInput = userInput + (char) ch;
         }
      }

      chatObject.setValue(userInput);
   // this is a local change.

      dsm.Write(oid,chatObject);
   // this will reflect the value to others.

      chatObject = dsm.Read(oid);
   // fetch current value from DSM (possibly changed by other chatters)
   }
   catch(Exception E)
```

```
    {
       System.out.println("Exception in client " + E.getMessage());
       E.printStackTrace();
    }
  }
}
```

The above program has created a shared object and made it sharable. The object (identified by its oid) can be accessed from a remote node by writing the following code. The assumption here is that the object identifier is known at the remote node. By using other operations such as Read, Remove in addition to the Read and Write shown here, complex distributed applications can be developed over the Internet, as the shared object space has been shown to be scalable. The programmer only has to write this client code and run it on the available machines. He/She must also run an OMR object per cluster and the DSM object in each node.

```
class accessSharedObject
{
  public static void main(String args[])
  {
   try {
      dclass chatObject = new dclass();
      Dsm dsm = new Dsm();

      System.out.println("value of args[0] is: " + args[0]);
   // assuming oid is args[0].

      chatObject = (dclass)dsm.Create(args[0],1);
      ....
   // code for getting user input, just as above.
      chatObject.setValue(userInput);
   // local write
      dsm.Write(args[0],chatObject);
   // reflects the value at all replicas
   }
   catch(Exception E)
   {
      System.out.println("Exception in client " + E.getMessage());
      E.printStackTrace();
   }
  }
}
```

ABOUT THE AUTHORS

A Vijay Srinivas obtained the MS (By Research) degree from the Distributed & Object Systems Lab, Indian Institute of Technology, Madras in 2001 and a Bachelors in Engineering from the University of Madras in 1998. He is currently finishing up his PhD from the same lab under Prof D Janakiram. His research interests span distributed systems, object technology and software engineering. His thesis is focused on building a data management platform for Peer-to-Peer grids. He is a student member of the IEEE and a member of the ACM. He can be reached at avs@cs.iitm.ernet.in. See also www.cs.iitm.ernet.in/∼ *avs*

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 prinicipal 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*