

# Policy-Based Autonomic Replication for Next Generation Network Management Systems

Cormac Doherty and Neil Hurley  
School of Computer Science & Informatics,  
University College Dublin, Ireland.  
Email: {cormac.doherty, neil.hurley}@ucd.ie

**Abstract**—We present a system for policy-based autonomic replication of data in the next generation of network management systems. The system supports multiple distinct replication schemes for a single data item in order to account for and exploit the range of consistency and quality of service requirements of clients. Based on traffic mix and client requirements, nodes in the system may make independent, integrated replica management decisions based on a partial view of the network. A policy based control mechanism is used to administer, manage, and control dynamic replication and access to resources.

We demonstrate the benefits offered to the next generation of distributed network management systems through use of such a system.

## I. INTRODUCTION

Due to the trend towards ubiquitous computing environments, customers of future networks are expected to use several separate devices, move between locations, networks and network types, and access a variety of services and content from a multitude of service providers. In order to support this multiplication of devices, locations, content, services and obligatory inter-network cooperation, there will be an increase in the scale, complexity and heterogeneity of the underlying access and core networks. Moreover, as a result of this “always online” lifestyle and the increased size and complexity of networks, there will be an increase in management and service related data by several orders of magnitude.

As exemplified by the OSI reference model, the Simple Network Management Protocol (SNMP) management framework, and the Telecommunications Management Network (TMN) management framework, network management (NM) has thrived on either centralized or weakly distributed agent-manager solutions since the early 1990s [1]. However, the aforementioned increase in size, management complexity, and service requirements of future networks will cause these solutions to struggle

[2]. As a result of this, Network Management solutions are being driven towards autonomically controlled, distributed solutions. Distributed network management addresses some of the shortcomings of current NM solutions and offers scalable, flexible and robust solutions to the demands presented by future networks.

As an enabling technology for these distributed network management systems, we propose a distributed data layer to dissociate data access from physical location. Furthermore, since we believe that some of the challenges posed by future networks are actually data management challenges, we additionally propose to add to this data layer the responsibility of autonomically managing the replication lifecycle and several levels of consistency for each replicated item of data. Replication and consistency is managed using a policy based control mechanism. As we place no restriction on the users, composition or content of this distributed data layer we use the general term “client” to refer to any entity accessing data, “node” to refer to the hardware across which the data layer exists and “data item” to describe any datum managed by the layer.

This paper presents such a distributed data layer and compares preliminary experimental results with analytic approximations of the performance achieved for management requests under various static data replication schemes and an initial set of consistency mechanisms.

## II. REPLICATION

Replication affords the possibility of increased performance and robustness of client applications as well as a degree of failure transparency. The degree to which these advantages are experienced is dependent upon access patterns, the current state of the network and the applied replication schemes. A replication scheme describes how a particular data item is replicated, that is: the number of replicas, where those replicas are placed and the choice of an update protocol governing consistency.

Previous work has indicated that replication schemes impact significantly on performance of distributed systems in terms of both throughput and response times [3]. Indeed, a bad replication scheme can negatively impact performance and as such, may be worse than no replication at all.

#### A. Static Vs Dynamic Replication

Static replication is the term used to describe replication in systems where replication schemes are developed and applied at design time and remain unchanged until an administrator manually intervenes. For example, static replication is currently used in TMN style network management systems as a means of increasing availability. Configuration Management (CM) data hosted by Network Elements (NE)s is replicated to Operations Support System (OSS) nodes such as Element Managers (EMs) and Network Managers (NMs); non-transactional update mechanisms are employed to maintain consistency.

In a system where attributes of traffic and the network are both known and unchanging, such static schemes are entirely appropriate. However, developing the “correct” scheme is closely related to the NP-complete file assignment problem [4], and optimal solutions are therefore impractical to calculate. In more typical systems where variations in topology, routing [5] and changes in access patterns are experienced over time, these inflexible replication schemes are unsuitable. Changes in traffic patterns or client population may negate the benefit of the replication scheme. Thus, an administrator would be required to redevelop replication schemes if the benefits of replication are to be maintained. To highlight the inadequacies of static replication consider the difficulties, time and expertise involved in developing and maintaining replication schemes for CM data in future networks. The dynamism of future networks, coupled with scale and heterogeneity of the constituent core and access networks would prohibit static replication.

Dynamic replication accounts for the natural fluctuations in user traffic by autonomically altering the replication scheme of a data item based on the current state of the network, user behavior and some performance related metric. Replication schemes are developed, adjusted and applied to data items so as to maximise some objective function. Typically, this involves examining access patterns, in one form or another. A very simplistic view of this is presented in [6] where the replication manager adjusts replication such that the client generating the arrival stream is best satisfied as defined by the objective function used by the system. However, the fact that the

access pattern perceived by a data item is the product of an entire population of clients is effectively ignored in most dynamic replication systems. That is, the system autonomically generating and applying replication strategies treats the arrival stream to a data item as though it were generated by a single client. The system then attempts to generate a replication scheme to suit this pseudo client, see Figure 1. As such, the range of consistency and quality of service requirements of clients is not taken into account when developing replication schemes.

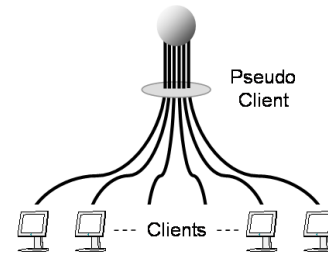


Fig. 1. Pseudo Client

For example, if a relatively small user group (Group A) requires strict consistency mechanisms to be enforced on a particular data item, the number of replicas in the system would ideally be kept to a minimum in order to minimise the overhead (time and messages) in updating; this would typically diminish the availability of the data item in question. Further, suppose another larger user group (Group B) is capable of operating with partially or temporarily inconsistent data. If the replication manager exploits the different requirements of different classes of client by allowing several levels of consistency for a single data item, requests from Group B could be satisfied using the set of replicas that are only periodically updated (thus reducing the “cost” of an update), while Requests from Group A could be satisfied by the smaller set of replicas that guarantee consistency.

Our work offers a novel addition to the area of dynamic replication and attempts to account for the various classes of client that contribute to the arrival stream experienced by a data item by taking advantage of the fact that some clients consistency requirements may not be as strict as others. As such, we propose to allow multiple distinct replication schemes for a single data item so as to best satisfy the requirements of all classes of client.

In order to provide this additional feature of dynamic replication we introduce policies to the system that must be enforced by all nodes in the network.

### III. POLICY-BASED AUTONOMIC REPLICATION

A policy is a set of high level rules to administer, manage, and control dynamic replication and access to resources. Policies allow specification of what the system should do rather than how it should be done. We apply policy based control to several aspects of the system.

In order to account for heterogeneity across nodes and restrict node resources available to the distributed data layer, we control the role a particular node plays in the network using a policy. Node policies are defined by an administrator and specify how a particular node can be used in terms of network, storage and processing resources. Node policies are used in determining which data items can or cannot be replicated locally.

```
<policy>
  <maxStorage>10Gb</maxStorage>
  <maxStoragePerDataItem>20Mb</maxStoragePerDataItem>
  <maxCPU>20%</maxCPU>
  <forwarding>redirect</forwarding>
  ...
  <minAccessRate>0.2TPS</minAccessRate>
  ...
</policy>
```

Fig. 2. An example node policy.

In addition, we introduce two levels of policy based control on replication. We associate, with each replica of a data item, a policy describing the level of consistency maintained by that replica and the performance metrics its host is prepared to maintain. As a means of controlling these *replica policies*, we associate with each logical data item a policy describing the limitations of all replica policies that can be applied to instances of a data item.

A *data item policy* specifies upper and lower bounds on performance metrics associated with accessing the data item and the minimum and maximum degree of consistency that must be maintained by any instance or replica of the data item, Figure 3.

When a data item is created, a policy controlling replication schemes that can be applied to instances of that data item is also created. The primary copy of the data item must adhere to the strictest set of requirements set out in the data item policy. We introduce this restriction to ensure there is at least one instance of the data item offering the highest level of consistency requirements and lowest response times. The primary copy of a data item with the policy depicted in Figure 3 would have to ensure a response time of at most 50ms for read messages and 100ms for write messages, and all updates must be processed immediately by the primary copy using Two Phase Commit.

We associate with each replica of a data item a policy

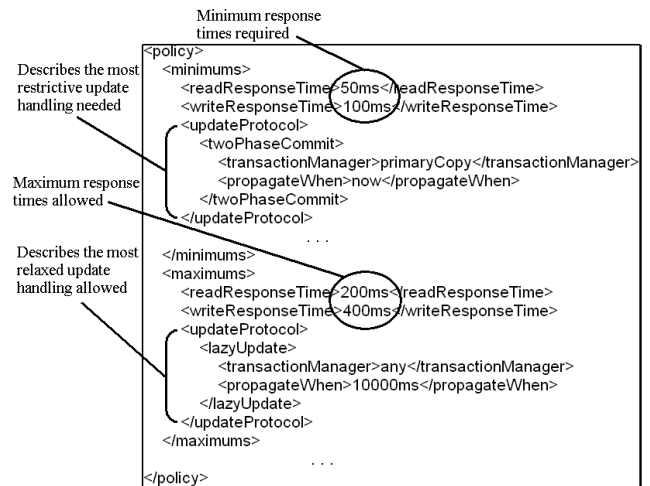


Fig. 3. An example data item policy describing maximum and minimum response times and consistency related mechanisms.

describing the level of consistency maintained by that replica and the performance metrics its host is prepared to maintain; these replica policies are controlled by the aforementioned data item policies. That is, the response times offered by a replica host must be less than the maximum response times specified by the data item policy and the update mechanism may not be such that it yields a level of consistency lower than that provided by the most relaxed consistency mechanism specified in the data item policy.

### IV. DISTRIBUTED DATA LAYER

#### A. Data Access

In order to dissociate data access from physical location, we introduce the replica lookup service. Each node hosting a replica of a data item may assign to that data item an arbitrary name. The replica lookup service is a federated database used to maintain mappings between these (arbitrary name, host name) pairs (Physical IDs) and a single identifier for all instances of the replicated data item, (Logical ID). The replica lookup service additionally provides access to these mappings. When a replica is moved, created or deleted the replica lookup service is updated accordingly. This service is based on the Gigggle Framework [7]; scalability of the framework has been demonstrated by the Replica Lookup Service (RLS) [8] of the Globus Toolkit. Applicability of a P2P extension to the RLS, P-RLS [9], is being investigated as a possible alternative. However, it should be noted that the RLS allows wildcard name matching while the P-RLS is restricted to exact name matching.

## B. Request handling

In order to describe how dynamic replication is performed and the effect of data item and replica policies, we first describe how requests are handled in the system.

Upon receiving a request from a client, a node,  $X$ , will extract from the request the identity of the relevant data item,  $d$ , and a set of consistency requirements,  $c$ , specified by the client in order to determine whether or not it is capable of satisfying the request. It is these client specified consistency requirements that determine a client's class. Capability to satisfy is determined by the data items stored locally by  $X$ , message type (read/write), policies associated with  $d$  and client consistency requirements  $c$ .

If an instance of  $d$ , with a replica policy that provides a degree of consistency greater than or equal to that specified by  $c$ , is found locally, a response will be sent directly to the client.

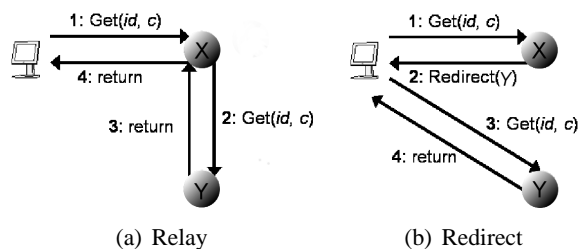


Fig. 4. Rerouting requests (relay Vs redirect)

If an instance of  $d$  is not stored locally or the replica policy provides a level of consistency lower than that specified by  $c$ , node  $X$  will query the replica lookup service to determine the location of another node in the network  $BY$  hosting an instance of  $d$  with a replica policy conforming to  $c$ . The client's request will then be either redirected or relayed to  $Y$ . If a request is relayed, node  $X$  will forward the request for  $d$  directly to  $Y$  and then pipe the response back to the client, Figure 4(a). Alternatively, if a request is redirected, node  $X$  replies directly to the client indicating  $Y$  should be contacted in order to satisfy the request for  $d$  with consistency requirements  $c$ , Figure 4(b).

Though relaying a request consumes resources on two nodes and consequently impacts on system throughput, relaying of requests serves two purposes within the data layer:

- (i) A node  $X$  may “piggy back” a request to host message on a request for  $d$  that is to be relayed to another node  $Y$ . Node  $Y$  makes a record of this request and may fulfill the request at a later date if a new replica of  $d$  is to be created.

- (ii) In order to avoid the extraneous bandwidth usage associated with determining network conditions, we passively monitor the transfer rate between nodes when a request is relayed.

If a node  $X$  finds itself having to forward a client request to a previously unknown node  $Y$ , node  $X$  will elect to relay the request to  $Y$  and indicate that it is doing so. In this way, both node  $X$  and node  $Y$  determine the bandwidth of the path between nodes  $X$  and  $Y$ . Nodes periodically relay client requests in order to maintain an accurate indication of bandwidth. This bandwidth information is used when a node is determining where to place a new replica, see Section IV-D for further details.

If the request is of type write, then the policy associated with the referenced data item is examined to determine how updates should be handled.

Regardless of the request type or whether or not a request was satisfied, each node maintains a temporary record of all requests received and how long it took to service those requests. As described in the following section, this record of requests is used to trigger modifications to replication schemes.

## C. Dynamic replication

In an attempt to increase scalability and avoid a single point of failure, we allow each node to make independent, integrated, replica management decisions based on a partial view of the system. In this way, each node shares the responsibility of replica management, performance monitoring and logging and autonomically adapts to their environment without the need for administrator intervention or centralised control. A node may change or attempt to change a replication scheme for a data item  $d$  for one of four reasons: (i) if it is unable to satisfy performance requirements associated with  $d$  (ii) if it finds itself redirecting requests for a data item it is prepared to host, (iii) if a resource usage boundary specified by a node policy is being broken, e.g. cpu usage, or (iv) if a locally hosted data item is unused. We now describe an example of each scenario and how a node alters the replication scheme in each case.

Upon realising its inability to satisfy performance requirements set out by either a data item or replica policy, a node will alter the replication scheme of a locally hosted data item. How the replication scheme is altered is dependent on the performance requirement being missed.

If a node  $X$  is unable to satisfy write related performance requirements for a data item  $d$ , one of the following actions will be taken:

- If the traffic mix experienced by  $X$  is such that the majority of read requests for  $d$  could be satisfied by a more relaxed degree of consistency, it will “downgrade” the replica policy for the locally hosted data item  $d$  so as to reduce the volume of write requests it receives and its commitment to maintaining consistency. A nodes ability to downgrade a replica policy is restricted by the node and data item policy.
- If  $X$  cannot alter the replica policy of the locally hosted data item  $d$  or altering the replica policy does not alleviate the problem,  $X$  will attempt to move or replicate  $d$  to another node  $Y$  using the replica placement algorithm described in Section IV-D.
- If  $X$  cannot rectify the problem by altering the replica policy of  $d$ , and the existence of a new replica host does not rectify the policy infringement, data item  $d$  will be removed from  $X$  provided it is not the primary copy.

A node  $X$  may attempt to change the replication scheme of a data item  $d$  it does not host if  $X$  redirects more requests for  $d$  than it satisfies for the least popular locally hosted data item. This attempt to alter the replication scheme is performed by “piggy backing” a *request to host* message on a relayed client request, Figure 4(a). Similarly, a node  $X$  may attempt to change the replication scheme of a locally hosted data item  $d$  if it redirects more read requests for  $d$  than it satisfies locally due to an inappropriate level of consistency. In such a scenario,  $X$  will attempt to “upgrade” the replica policy for the local copy of  $d$  such that it provides a level of consistency amenable to a majority of the redirected requests. The replica policy is upgraded on condition that the node policy permits it and the last replica management operation on the replica policy was not a downgrade.

Finally, in order to maintain efficient usage of node resources and avoid the overhead of maintaining consistency across unused replicas, nodes may attempt to remove a locally hosted data item. That is, once the arrival rate for a locally hosted data item  $d$  falls below a threshold defined in the node policy,  $d$  will be removed. Similarly, if a node cannot satisfy requirements for a data item  $d$  by modifying the replication strategy or creating a new replica,  $d$  will be removed. Locally hosted data items are removed provided they are not the primary copy.

#### D. Replica Placement

As mentioned above, the onus of replica management is distributed across all nodes in the system; when a

node recognises its own inability to meet performance requirements we require that node to take action to remedy the situation. A node may alter a replica policy within the boundaries set out by the relevant data item policy or it may choose to create a new replica on another node in the system. We now describe how a node selects where to place a new replica.

As detailed in Section IV-B and Section IV-C, each node maintains a record of: (i) the bandwidth experienced between itself and nodes it has had contact with (*bpsRecord*), and (ii) nodes that have sent *request to host* messages (*rthRecord*). These *request to host* messages expire and are removed from the *rthRecord* based on the earlier of two expiration dates; one optionally specified by the request originator and one associated with each node. A node sending a *request to host* may specify an expiration time  $t$  beyond which the request should be considered invalid and all nodes are configured such that received *request to host* messages may not have a life span greater than  $t'$ . Thus, a *request to host* message is invalidated and removed from the *rthRecord* after  $\min(t', t)$ .

Combined, these records form a list of known hosts that is used in determining where to place a new replica (*knownHosts*). The algorithm a node uses to determine where to place a replica of a data item  $d$  is shown below:

---

#### Algorithm 1 placeReplica( $d$ )

---

```

1: knownHosts  $\leftarrow$  bpsRecord + rthRecord $d$ 
2: nodeList  $\leftarrow$  sortBy(knownHosts,  $d$ , bps)
   # sorts nodes in knownHosts on the age of request
   # to host (rth) messages and then on bandwidth
3: hostFound  $\leftarrow$  false
4: currentHost  $\leftarrow$  nextHost()
5: while ( currentHost  $\neq$  null ) do
6:   hostFound  $\leftarrow$  create( $d$ , currentHost)
7:   currentHost  $\leftarrow$  nextHost()
8: end while
9: if ( hostFound = false ) then
10:   forceCreate( $d$ , selectRandomHost())
11: end if

```

---

If a node  $X$  has received *request to host* messages for the data item to be replicated  $d$ , the node with the highest bandwidth that sent the oldest *request to host* messages is selected as the new replica host. If no *request to host* messages have been received, the known host with the highest bandwidth is chosen as the new host. Since a node  $Y$  may refuse to host a replica of  $d$  because its *node policy* forbids it,  $X$  will continue to try nodes in

*nodeList*, until one accepts or until all nodes have been tried. If all nodes in *nodeList* refuse to host a new replica of  $d$ , one is selected at random and forced to host  $d$ . In order to avoid violating policies, the new replica host  $Y$  may delay receipt of the data item in order to adjust the set of locally hosted data items to facilitate hosting  $d$ . Note that this may require a run of Algorithm 1 at  $Y$ .

## V. ANALYTIC MODEL

The purpose of the analytic model presented in [3] is to determine maximum network throughput through analysis of message flows under different replication schemes, thus enabling the prediction of optimum replication schemes.

The network is modeled as a set of  $n$  distributed nodes, partitioned into  $K$  different node types with  $n_i$  nodes of each type  $1 \leq i \leq K$ . Connectivity is described by an  $n \times n$  adjacency matrix  $X$  where the  $n_i \times n_j$  sub-matrix  $A_{ij}$  describes the connections between nodes of type  $i$  and nodes of type  $j$ . We consider  $\tau$  message classes, each of which can be handled by a single node. Messages arriving to a node may be handled locally, or may need to be redirected to a remote node that can handle them.

The following parameters are used to describe the flow of messages through the system:

- $a_{uv}$  an element of the sub matrix  $A_{ij}$  indicating whether or not node  $u$  is connected to  $v$ .
- $\ell_i$ ,  $i = 1, \dots, K$  is the probability that a message can be handled locally without replication;
- $\beta_{ij}$  a  $K \times K$  matrix representing the probability that data on a node of type  $i$  is replicated on a node of type  $j$ . That is if  $u$  is a node of type  $i$  and  $v$  is a node of type  $j$  and  $a_{vu} = 1$ , then the probability that primary data on  $v$  is replicated on  $u$  is  $\beta_{ij}$
- $\hat{\beta}_{ij}$  a  $K \times K$  matrix representing the probability that data on a node of type  $i$  is replicated on a node of type  $j$ , given that a message arriving to a node of type  $i$  is for data on a node of type  $j$
- $r(s)$  a characteristic function used to assert whether a message can be handled using replicated data. If  $r(s) = 1$  then the message can be handled by a replica, otherwise  $r(s) = 0$ .

The arrival rate of messages of a particular type  $s$  to nodes of type  $i$ ,  $\lambda_{is}^0$ , is found to be a combination of: messages arriving from outside the network  $\lambda_{\text{EXT}}$ , messages rerouted from other nodes  $\lambda_{\text{RR}}$ , and update propagation messages  $\lambda_{\text{UP}}$  from other nodes.

$$\begin{aligned} \lambda_{\text{EXT}} &= \lambda_{is}^0 (\ell_i + r(s)(1 - \ell_i) \sum_j^K (\rho_{ij} \hat{\beta}_{ji} \sum_v a_{vu})) \\ \lambda_{\text{RR}} &= \sum_{j \neq i}^K (\lambda_{js}^0 (1 - \ell_j) \rho_{ji} (1 - r(s) \hat{\beta}_{ij} a_{vu})) \end{aligned}$$

$$\begin{aligned} \gamma_{i,s} &= \lambda_{\text{EXT}} + \lambda_{\text{RR}} \\ \lambda_{\text{UP}} &= (1 - r(s)) \sum_j \beta_{ji} \gamma_{j,s} \sum_v a_{vu} \\ \lambda_{is}^0 &= \lambda_{\text{EXT}} + \lambda_{\text{RR}} + \lambda_{\text{UP}} \end{aligned} \quad (1)$$

Using Equation 1, the arrival rate of messages to the system as a whole,  $\lambda^0$ , is found to be:  $\sum_s \sum_i n_i \lambda_{is}^0$ . Maximum throughput at a node occurs when utilisation is equal to 1. We determine the global arrival rate,  $\lambda^0$ , which yields the maximum throughput for each node type. Maximum system throughput is then taken as the arrival rate, which causes one or more node types to be fully utilised.

For a full explanation of this model, its parameters and derivation of equations, we refer the reader to [3].

## VI. RESULTS & ANALYSIS

We are interested in demonstrating how the use of multiple distinct update mechanisms offering different degrees of consistency potentially offers an increase in throughput to NMSs and how removing the hierarchical node relationships of traditional TMN-style NM further increases throughput of the entire system.

### A. Experimental Setup

In order to demonstrate and observe the effect of multiple consistency mechanisms independently of the level of distribution, we first compare preliminary experimental results to model predictions for a hierarchical system representative of current network management solutions. The system under study is a 2-layer network of OSS and Radio Network Subsystem (RNS) nodes, Figure 5. Though this system is scale-limited, we believe it to be sufficient for demonstrating preliminary work; larger scale experiments will undertaken in the future.

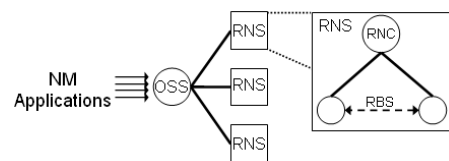


Fig. 5.

Messages arrive at OSS nodes from management applications and are either handled locally using replicated RNS data or are re-routed to an appropriate RNS node. All messages request primary data on RNS nodes. This comparison also serves as means of verifying results of both the model and the system under development.

We apply two contrasting push-based update propagation mechanisms, namely: *Two Phase Commit* and

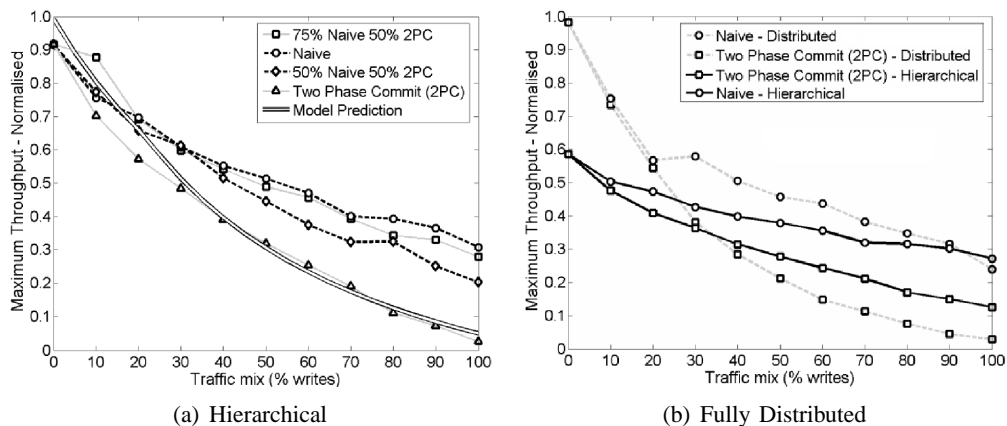


Fig. 6. Maximum System Throughput of Client requests - 60% of primary data is replicated to 100% of possible replica hosts.

*Naive*. Two Phase Commit allows all nodes hosting a replica of a data item (cohorts) to agree to perform an update. Two Phase Commit realises ACID properties in the absence of network or node failure by locking data items, employing a single point of control (coordinator), at the expense of multiple coordination messages per update. The two phases are the commit-request phase, in which the coordinator attempts to prepare all the cohorts, and the commit phase, in which the coordinator completes the update at all cohorts. We assign the role of “coordinator” to the node hosting the primary copy of a data item. Naive update handling is another master–slave form of replication. Naive again employs the node hosting the primary copy to propagate updates but “naively” assumes no errors, complications or conflicts will occur and as such refrains from expensive distributed locking mechanisms and coordination messages.

In comparisons with measurements taken from live networks and test sites, the model’s throughput predictions were found to have a maximum error of approximately 14%. In order to avoid publishing exact performance figures, we have normalised all throughput figures. The model has been configured such that the response time for update messages is representative of update handling using two phase commit.

### B. Results

In comparing throughput predicted by the model (*Model Prediction*) to experimental results where update handling is performed using 2PC, we find the model’s prediction of throughput to be marginally over-optimistic for read–dominant traffic mixes ( $< 30\%$  writes). Conversely, traffic mixes consisting of  $> 30\%$  writes, model predictions are markedly accurate.

Figure 6(a) also presents throughput figures in the

same system when alternate update handling techniques are applied. As the more relaxed form of update handling consistency, *Naive* offers the highest system throughput due to the minimal cost of updates. Predictably, we find a mix of both *Naive* and 2PC updates offers higher throughput than 2PC alone. However, the increase in throughput experienced is dependent upon the ratio of Naive to 2PC updates. Since the response time of an update using 2PC dominates the response time of a Naive update, we find the relationship between throughput and the proportion of updates using 2PC to be indirectly proportional. That is, a mix of 50% updates using 2PC and 50% Naive updates does not equate to a throughput midway between that achieved using only Naive updates and that using only 2PC, see Figure 6(a).

### C. Extent of Distribution

Figure 6(b) depicts maximum throughput figures in a system where the hierarchical node relationships of current NM techniques are removed. In this fully distributed system, all nodes play an equal role in the distributed data layer. Note that the increased degree of distribution results in an increase in the number of replicas of a replicated data item, and thus an increase the number of replicas that (i) may be used to satisfy a read and (ii) need to be updated.

When traffic mix consists of less than  $\sim 30\%$  updates, an increased degree of distribution yields a notable increase in throughput. As the traffic mix approaches 30% updates, the relatively high cost of updates in comparison to reads begins to dominate and throughput is limited accordingly.

Since the response time of an update using Naive update handling is small in comparison to the response time of 2PC, system throughput when Naive update

handling is applied is greater than that achieved using 2PC across all traffic mixes.

Finally, as the number of replicas to be updated increases with the degree of distribution, we see the throughput of updates using 2PC in the distributed system fall below that achieved using 2PC in the hierarchical system. This feature of the graph is evidence of the high cost of maintaining strict consistency across many replicas in a distributed system. Conversely, the temporally inexpensive Naive update technique permits high throughput even for update rich traffic mixes.

#### D. Evaluation

By applying multiple update mechanisms and an increased degree of distribution we have demonstrated the potential performance gains to be achieved through use of a distributed data layer. By combining multiple update mechanisms simultaneously and adapting the associated policies autonomically, see Section IV, throughput can be maximised across all traffic mixes. For example, in Figure 6(b), when traffic mix consists of less than 20% updates, replica policies can be such that all replicas are updated using 2PC. However, as the mix exceeds 30% updates, the high cost of maintaining consistency across many replicas becomes apparent and throughput falls below that achieved in a hierarchical system. If replica policies were to be adapted such that a portion of replicas were to be updated using Naive update handling, the high cost of an update would be reduced and throughput would increase accordingly. Thus, multiple distinct replication schemes for a single data item offers increased performance.

### VII. CONCLUSIONS

As the pervasive networks of the future are developed, a scalability crisis looms in the current network operations and maintenance (OAM) infrastructure. The increased complexity and heterogeneity in terms of both equipment vendors and access network technologies is driving network management research towards more distributed architectures. We have presented a distributed data layer as an enabling technology for these distributed network management systems. Through autonomic replication and distribution of data, this distributed data layer offers the possibility of increased scalability of client applications by dynamically altering replication schemes based on network state. We employ policy based control mechanisms on both nodes and data in the network.

The task of managing replication in the system is distributed across all nodes. Each node makes independent

decisions based on a partial view of the system. In this way we eliminate the need for centralized management, control and logging and provide a system with no single point of failure that adapts to its working environment.

In addition we contribute to the field of dynamic replication. Current dynamic replication systems treat the arrival stream to a data item as though it were generated by a single client and attempt to generate replication schemes to suit this agglomerative client. We present an alternative to this notion of dynamic replication and propose a means of applying several distinct replication schemes and thus several objective functions to a single data item using policy based control.

### REFERENCES

- [1] J.-P. Martin-Flatin, S. Znaty, and J.-P. Hubaux, "A Survey of Distributed Enterprise Network and Systems Management Paradigms." *Journal of Network and Systems Management*, vol. 7, no. 1, 1999.
- [2] M. Burgess and G. Canright, "Scalability of Peer Configuration Management in Partially Reliable and ad hoc Networks," in *Proceedings of the 7<sup>th</sup> IFIP/IEEE Conference on Network Management*, vol. 246, IFIP/IEEE. Colorado Springs, USA: Kluwer, March 2003, pp. 293–305.
- [3] N. Hurley, C. Doherty, and R. Brennan, "Modelling Distributed Data Access for a Grid-Based Network Management System," in *Proceedings of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Atlanta, USA, September 2005.
- [4] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, vol. 14, no. 2, pp. 287–313, 1982.
- [5] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet Routing Instability," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM Press, 1997, pp. 115–126.
- [6] V. Duvvuri, P. J. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web." *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1266–1276, 2003.
- [7] A. L. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, R. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Press, 2002, pp. 1–17.
- [8] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, "Performance and Scalability of a Replica Location Service," in *Proceedings of the 13th International Symposium on High-Performance Distributed Computing*. IEEE Computer Society Press, June 2004, pp. 182–191.
- [9] M. Cai, A. Chervenak, and M. Frank, "A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society Press, November 2004, p. 56.