Autonomic Distributed Data Management with Update Accesses

Cormac Doherty
Department of Computer Science & Informatics
University College Dublin
Dublin, Ireland
cormac.doherty@ucd.ie

Neil Hurley
Department of Computer Science & Informatics
University College Dublin
Dublin, Ireland
neil.hurley@ucd.ie

ABSTRACT

This paper discusses and motivates autonomic replication in a data management layer for network management systems. The data management layer simultaneously supports multiple update protocols for single logical data–item, allowing different levels of performance and consistency to be offered to different groups of users. How replication is applied is key to maximising system performance. The challenge lies in the fact that the optimal scheme depends on dynamic characteristics of the workload. Autonomically reconfiguring replication in response to the changing workload using feedback control.

The goal of the paper is to demonstrate the applicability of feedback control to maintaining a given performance metric in a changing environment. The presented work is intended as an investigation and proof of concept into the feasibility of autonomically managing a facet of replication: specifically, the number of replicas. Replica placement and optimality of management decisions is currently considered out of scope and part of future work.

1. INTRODUCTION

Within an information system, replication affords a wide range of advantages ranging from increased robustness of client applications to a degree of failure transparency. The degree to which any of these advantages are experienced is dependent upon access patterns, the current system and network state, the data being replicated, 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 update propagation technique governing consistency. There exists two techniques to development and application of these replication schemes, namely static and 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 largely unchanged until an administrator manually intervenes. In a system where attributes of traffic and the network are both known and unchanging, such static schemes are entirely appropriate. In fact, experience and in-depth knowledge has allowed administrators to typically outperform software based solutions in static environments. In more typical systems where variations in topology, routing [14] 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 applied replication scheme or even damage performance. Prompt and frequent redevelopment of replication schemes by an administrator is required if static replication is to be applied in a dynamic environment. For large scale highly dynamic systems, administrator control of replication is not only susceptible to expensive human error but represents a non–trivial cost in terms of time, expertise and money. These costs are additionally inflated when response time is considered. Changing network conditions reducing the efficacy of a given replication scheme may cause an administrator to be notified; by the time such a notification has been received, the problem has been identified and a solution formed, the transient network conditions may have progressed to a new problem where the derived solution does not apply, or regressed to 'normal' where the derived solution no-longer needs to be applied. The problem with this is two–fold: 1) QoS and performance conceivably deteriorated during the problem period which may incur financial penalties as a result of SLAs and, 2) application of a solution to a transient problem that has since passed creates a new problem.

Dynamic replication accounts for the natural fluctuations in user traffic and behaviour by adapting replication schemes based on the current state of the network, user behaviour and some target performance related metric(s). Typically this is achieved through the introduction of an element of context awareness. A system employing dynamic replication will monitor its environment to determine, how and when to alter replication schemes, and how these alterations effect the system. In this way, replication schemes are developed, adjusted and applied so as to maximise some objective function representing performance[10]. The arguments and disadvantages to administrator controlled replication presented above could arguably be cited here as well. However, the proposition is to autonomically, as opposed to automatically, manage replication. That is, allow nodes to modify replication schemes to satisfy local goals and handle received traffic by altering their own role in the system and cooperating with other nodes.Âă Not only does this autonomy decentralise replica management but additionally introduces

Autonomics October 28-30, Rome, Italy.

an element of *divide and conquer* where individual nodes handle small, local, problems as opposed to a system wide problem. Furthermore, as replication is in essence a function of the environment, schemes are modified as and when environmental changes.

The work presented herein is concerned with autonomic replication and aims to maintain QoS and Service Level Agreements (SLAs) in a dynamic distributed environment. Feedback control is applied to determine how best to adapt replication schemes so as to maintain QoS/SLAs.

The remainder of this paper is structured as follows. Section 2 describes the the context in which our work is applied. Section 3 introduces the distributed data management system and its relevant features. Section 4 defines autonomic replica management in the context of our work, develops a feedback control mechanism to regulate average message response through changes in replication and presents some preliminary results. Finally, Section 6 presents conclusions and future work.

2. MOTIVATION

As 3G mobile networks are deployed, and pervasive, highly heterogenous 4G networks are developed, a scalability crisis looms in the current network operations and maintenance (OAM) infrastructure. 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. Furthermore, based in part on the 2G to 3G experience, an explosive growth in the number of network elements (NEs) to be managed is predicted. Each additional NE, type of NE, and inter-working function between different access network technologies, adds to the volume of management data that must be collected, queried, sorted, stored and manipulated by OAM systems. 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 centralised or weakly distributed agent-manager solutions since the early 1990s[15]. However, the aforementioned increase in size, management complexity, and service requirements of future networks will present challenging non-functional requirements that must be addressed in order to deliver scalable OAM data management sub-systems[2]. More distributed architectures for next generation OSS platforms are one approach to providing scalable, flexible and robust solutions to the demands presented by future networks.

Within a typical NM solution management actions depend on, and are in essence issued by, the Network Management System (NMS); herein lies the failure of centralised NM to cope with the scale and complexity of future networks. By distributing the responsibility of NM, several potential advantages may be achieved: network traffic and load on the NMS can be reduced by performing processing on, or close to, NEs; scalability is improved as capacity to perform NM grows with the network and distributed NM is inherently more robust. Furthermore, as the compute power of NEs grows, the opportunity to increase the degree of distribution of NM also increases and thus reinforces the aforementioned advantages.

These aspects of future networks are driving NM solutions towards autonomically controlled, distributed solutions [2, 20]. Distributed NM addresses some of the shortcomings of current solutions and offers scalable, flexible and robust solutions to the demands presented by future networks.

3. DISTRIBUTED DATA LAYER

As an enabling technology for these distributed NM systems, a distributed data layer to dissociate data access from physical location is proposed [8, 7]. Furthermore, as some of the challenges posed by future networks are data management challenges, the responsibility of autonomically managing the replication life-cycle and several levels of consistency for each replicated item of data is bestowed upon the data layer. Replication and consistency are managed using a policy based control mechanism. As no restriction is placed on users, composition or content of the distributed data layer, the general term "client" is used to refer to any entity accessing data, "node" to refer to the hardware across which the data layer exists and "data item" to describe a logical datum managed by the layer.

A fundamental observation motivating this work is the fact that the access pattern perceived by a data item is the product of an entire population of clients. This observation is not exploited in most dynamic replication systems. That is, a system applying replication treats the arrival stream to a data item as though it were generated by a single client and attempts to generate a "one size fits all" replication scheme to suit this pseudo-client. As such, the range of consistency and quality of service requirements of all clients contributing to an arrival stream is not taken into account when applying replication, see Figure 1. 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 propagating updates; 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 differing 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.

In order to account for and exploit the various classes of client that contribute to the arrival stream experienced by a data item, multiple distinct replication schemes are simultaneously applied to a single data item so as to best satisfy the requirements of all classes of client. To provide this additional feature of dynamic replication policies are introduced to the system that must be enforced by all nodes.

3.1 Policy Based Control

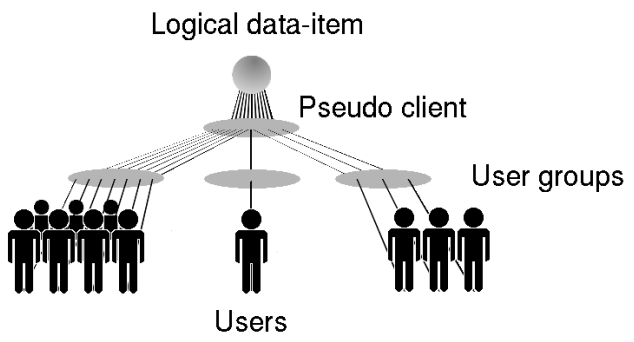

Figure 1: Exploiting QoS & Consistency Requirements

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 as opposed to 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 resources consumed by the distributed data layer, the role a particular node plays in the network is controlled 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 be replicated on a specific node.

```
<policy type="node">
        <maxStorage unit="Gb">10</maxStorage>
        <maxStoragePerDataItem unit="Mb">20</maxStoragePerDataItem>
        <maxCPU>0.2</maxCPU>
        <forwarding>redirect</forwarding>
                *       *       *
        <maxQueueLength>300</maxQueueLength>
        <poolSize>10</poolSize>
</policy>
```

Figure 2: Example Node Policy.

A node governed by the policy depicted in Figure 2 is restricted in terms of the role played within the system such that it may not: use more than 10Gb of local storage, host a single data item larger than 20Mb, use more than 20% of CPU time; additional node configuration parameters impacting performance are also specified. As a node policy is only pertinent to the node to which it applies, it is stored by and accessible to only that node.

Two data centric policies are used to control replication. A *data item policy* specifies upper and lower bounds on consistency related parameters and performance metrics that must be maintained by any replica of the data item to which the policy refers, Figure 3. Associated with an instance of a logical data item is a *replica policy* indirectly describing the level of consistency maintained by that replica and request related performance metrics its host is prepared to maintain; these replica policies are bounded by data item policies.

The policy depicted in Figure 3 defines the boundaries for all replica policies associated with the data item identified by the GUID (Globally Unique IDentifier) and may be decomposed into three sections. The first section, *metrics*, specifies minimum and maximum request related performance metrics that must be maintained by a replica host. Examples

```
<policy guid="00ee0a4a-6709-1029-8a16-00087427f74a" type="data" >
   <metrics>
      <writeServiceTime units="ms" min="100" max="200" />
      <readServiceTime units="ms" min="50" max="100" />
      <replicaCount min="10" />
            *    *    *
   </metrics>
   <triggers>
      <trigger_def id="01" >
         <name>count</name>
         <param min="5" max="25" />
      </trigger_def>
      <trigger_def id="02" >
         <name>composite</name>
         <trigger_def id="021" >
            <name>count</name>
            <param min="10" max="100" />
         </trigger_def>
         <trigger_def id="022" >
            <name>time</name>
            <param min="10" max="100" />
         </trigger_def>
      </trigger_def>
         *    *    *
   </triggers>
   <protocols>
      <protocol>
         <name>TPC</name>
         <param id="1" >primary</param>
      </protocol>
      <protocol>
         <name>TSAE</name>
         <param id="1" >any</param>
         <triggers>01 02</triggers>
      </protocol>
          *    *    *
   </protocols>
    </policy>
```

Figure 3: Example Data Item Policy.

of such metrics include minimum number of replicas (availability) and service time. Within the context of a data item policy, maximum service times are considered hard limits in that no descendent replica policy may advertise a greater service time. Conversely, minimum service times are soft limits; they state the lowest service time required by the most demanding class of client. The second section defines a set of applicable *triggers* and parameter boundaries. The last section defines update protocols that may be used and the set of triggers that may be applied to those protocols. Current possibilities include Time Stamped Anti Entropy (TSAE), Gossipping, and Two-Phase commit (2PC).

Within the context of our work we use the general term *trigger* to refer to the means by which update propagation is initiated. Zero or more triggers may be applied to any update protocol. The following briefly describes some of the possible triggers applicable within the system. *Count* triggers allow update propagation to be initiated when a specified number of tentative updates have been received. Two time based triggers exist, *time-push* propagates updates every n ms, and *time-pull* allows a remote replica to 'pull down' updates when it has not received updates in the last m ms. *Content* triggers initiate propagation as a function of the changes made to a data item. Finally, a *composite* trigger allows the disjunction (or conjunction) of multiple triggers.

When a data item is created, a *data item policy* controlling replication schemes that can be applied to instances of that data item is also created and installed in the replica location service. The creator of a data item is responsible for defining a data item policy. The primary copy of the data item must adhere to the strictest set of requirements set out in the data item policy. This restriction ensures there is at least one instance of the data item governed by the most stringent policy, regardless of its cost.

A replica policy defines a particular point in the param-

eter space defined by a data item policy. For example, the *replica policy* of the primary copy of a data item with the policy depicted in Figure 3 must ensure a response time of at most $50ms$ for read requests and $100ms$ for update requests, all updates must be processed by the primary copy using Two Phase Commit and triggered update processing is disallowed. In contrast to data item policies, within the context of replica policies, performance metrics are considered soft limits as breaking of a performance metric triggers alteration of a replication scheme. Both data item policies and replica policies are installed in a federated database based on the Replica Location Service (RLS) of the Globus toolkit but are additionally cached by hosting nodes, [3, 4].

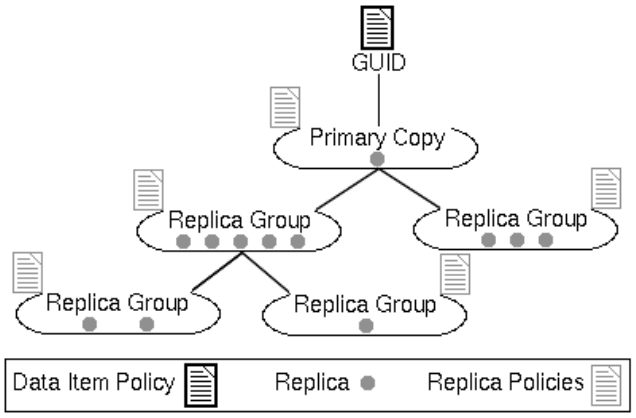

Figure 4: Replica Hierarchy

Replicas associated with a particular *replica policy* form a replica group, where consistency *within* a group is governed by the replica policy. Excluding the replica group to which the primary copy belongs, each replica group has a parent replica group and zero or more child groups. These replica groups are organised into a logical hierarchy. This hierarchy indirectly groups nodes hosting replicas based on the group to which a policy belongs. Figure 4 shows the replication hierarchy for a single logical data item. The root replica group contains a single replica (primary copy) and offers the most stringent set of requirements defined by the data item policy. The root policy has two child groups, each of which is governed by a less stringent replica policy than that governing the root group. As a child group receives update requests after its parent, each group necessarily offers a less consistent view of the data item than the root group. Thus, consistency within a group is determined solely by the replica policy governing the group, and consistency of a group is determined by its ancestors.

3.2 Data Structures

Each node in the system maintains a record of: (i) the bandwidth experienced between itself and nodes it has had contact with (*bpsRecord*), (ii) nodes it frequently relays requests to due to an inability to satisfy them locally, and (iii) nodes that have sent *request to host* messages (*rthRecord*). Receipt of a *request to host* message signifies a nodes interest in hosting a replica of data item it is receiving requests for but is unable to satisfy, see (ii). Combined, these records form a list of known hosts. Though intended only as a place–holder for future work, a trivial algorithm using these data structures has been developed to handle replica placement [7].

Though replication and distributed data management offer advantages to distributed NM in future networks, the complexity of managing such a system is comparable to that of the NM system it supports. Active creation and maintenance of replicas and replication schemes by administrators is possible, the necessary expertise and time involved in 'optimally' administering a system of the expected complexity and magnitude represents a non-trivial cost. We apply autonomic control mechanisms, such that the system can self-monitor and adjust policies so that the QoS/SLA constraints are maintained.

4. AUTONOMIC REPLICA MANAGEMENT

Replication affords the possibility of increased 'performance' and robustness of client applications as well as a degree of failure transparency. The appropriate measure of 'performance' is subjective with respect to the system; it may relate to system–wide characteristics such as response time, throughput or utilisation of nodes in a distributed database management scenario; or in a grid environment, replication may be motivated by the high likelihood of node failure and quantified by availability. Our work specialises to the first scenario. We are interested in maintaining a desired level of performance, measured in terms of throughput or response time, as well as data consistency, under changing workload conditions.

In the context of the state-of-the-art, our problem area is interesting in that we wish to support update accesses as well as read accesses. Replication necessarily requires updates to be propagated to all replicas, an overhead not incurred in read-only systems. However, for optimal performance, update propagation will not necessarily be performed as an atomic operation for every update, thus we allow replicas to be inconsistent with the most up-to-date instance for some period of time and will trade some tolerance to inconsistency for shorter response times. The challenge then is to monitor the conditions under which the system is operating, and modify the replication strategy when system performance falls outside a desired tolerance.

In an attempt to increase scalability, avoid a single point of failure, and increase autonomy, each node makes independent, integrated, replica management decisions based on a partial view of the system. That is, each node shares the responsibility of replica management, performance monitoring and logging and autonomically adapts to its environment, thus precluding the need for administrator intervention or centralised control. We are interested in autonomically reconfiguring replication schemes in response to changing workload using feedback control.

4.1 Feedback Control

Feedback control aims to manipulate measured output or output characteristics of a system through alteration of control inputs. The key component in a feedback control loop is the *controller*, which is responsible for comparing the *reference input*, y_r, to the *measured output*, y, and altering *control inputs*, u, such that the difference between y and y_r, the *control error*, e, is minimised, that is, $y \cong y_r$.

Though y could conceivable represent any of the perfor-

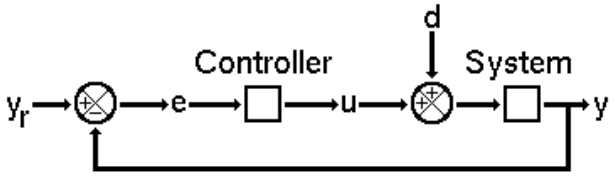

Figure 5: Feedback control loop

mance metrics specified in the policies described in Section 3.1, the current emphasis of our research is dynamic replication. Thus, y is restricted to those metrics specified in data centric polices. Similarly, u could represent various aspects or parameters of the system but is limited to those concerning replication schemes. The reference input, y_r, is determined by the desired QoS metrics specified in the data centric policies. We use average message response time (MRT), as an output signal and the percentage of logical data items fully replicated (REP) as the input signal. In particular, if a data item is replicated, then a replica is stored on every node. Changes in traffic mix are modelled as disturbances, d to the system.

Conceptually, the feedback control loop works as follows: the desired average response time, y_r, for a particular replica group (see Section 3.1) is specified as part of the replica policy for that group. At the end of every sampling period the controller compares the measured output signal y to y_r and produces a control signal, u, representing the change in REP needed to correct the difference, e, between y and y_r.

4.2 Controller Design

This section describes production of the control signal.

Let $y(k)$ and $u(k)$ respectively denote the output and input signal in the k^{th} sampling period. Controller design begins with constructing a model of the system to quantify the relationship between control inputs and measured outputs. This model takes the form of a difference equation and relates current and past outputs to current and past inputs. We adopt a first–order model, though higher order models are possible the results presented in the following section demonstrate that a first–order model successfully models the system. The relationship between REP and MRT is modelled as a linear difference equation of the form:

$$y(k+1) = ay(k) + bu(k) \tag{1}$$

Where a and b are specific to the system being modelled and are estimated from training data collected from the system being modelled. Such training data consists of a set of $(y(i), u(k))$ pairs. Using least squares regression, Equation 1 for our system and the selected operating point, $\overline{\text{REP}} = 49$ and $\overline{\text{MRT}} = 4981$, is found to be:

$$y(k+1) = 0.73y(k) + 45.18u(k) \tag{2}$$

The next step is to determine and model how an input signal is transformed into an output signal by the system under study. This transformation is referred to as the transfer function of the system and can be found by taking the Z-transform of both sides of the difference equation (Equation 2), setting all initial conditions to zero, solving for $Y(z)$ in terms of $U(z)$ and computing the ratio of $Y(z)$ to $U(z)$, $G(z)$:

$$\sum_{k=0}^{\infty} y(k+1)z^{-k} = \sum_{k=0}^{\infty} ay(k)z^{-k} + \sum_{k=0}^{\infty} bu(k)z^{-k}$$
$$zY(z) - zY(0) = aY(z) + bU(z)$$

Setting initial conditions to zero:

$$zY(z) = aY(z) + bU(z)$$
$$zY(z) = 0.73Y(z) + 45.18U(z)$$
$$Y(z) = \frac{45.18U(z)}{(z - 0.73)}$$

Computing the ratio of $Y(z)$ to $U(z)$:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{45.18}{z - 0.73} \tag{3}$$

The transfer function has one pole (root) at $z = 0.73$ which is inside the unit circle, $|z| < 1$ and hence the model is BIBO stable. That is, for any bounded input ($0 \leq \text{REP} \leq 100$), the output, MRT, is also bounded. Steady–state gain characterises the effect of a constant input u_{ss} on steady state output, y_{ss} and is defined to be the ratio y_{ss}/u_{ss}. Steady–state gain is more commonly computed using the transfer function $G(z)$ and a unit input increase [11].

$$G(1) = \frac{y_{ss}}{u_{ss}} = \frac{0.73}{1 - 45.18} = -0.02$$

As the gain is negative, an increase of 1 in REP, will cause an decrease in MRT of 0.02. Note that as this is a linear model, accuracy of this relationship is greatest near the specified operating point, see Table 1.

The parameter b describes the relationship between the input REP and the output MRT, as $b > 0$, a positive correlation between REP and MRT is expected. That is, an increase in REP will lead to an increase in MRT. This conclusion is only valid within the context of this model and is in part due to the relative expense of write access when compared to read access. As a write access to a replicated data–item involves additional (propagation) writes to all replicas, the response time of a write access increases by a factor of 1 with every new replica. The reduced response time of read accesses with replication are dominated by the expense of write accesses.

The role of the controller as the component responsible for determining the control input u given the current and past control error e is now formalised through specification of a *control law*. A proportional control law is used in this model:

$$u(k) = K_p e(k) \tag{4}$$

where K_p is a constant referred to as the controller gain and $e(k)$ denotes the error signal in the k^{th} sampling period, i.e. $e(k) = y_r - y(k)$. Designing a proportional controller requires selection of K_p such that the closed loop system has the desired properties for stability, steadty–state gain and transient response. To assess these properties, the transfer function $F_r(z)$ is constructed.

Finally, to complete the controller design the closed–loop transfer function is found to be:

$$F_R = \frac{0.73K_p}{z - 45.18 + K_p 0.73} \tag{5}$$

k	1	2	$\cdots$	25	26	27	$\cdots$	50	51
$\tilde{\mathbf{u}}$**(k)**	0	2	$\cdots$	48	50	52	$\cdots$	98	100
$\tilde{\mathbf{y}}$**(k)**	176	263	$\cdots$	2297	2744	1261	$\cdots$	24166	28589

Table 1: Subset of data used in developing model.

Since the closed–loop function is first–order, there is only one closed–loop pole, $p_1 = 45.18 - 0.73K_p$. In order to ensure a stable system, it must be ensured that this pole lies within the unit circle $|45.18 - 0.73K_p| < 1$, thus we solve for these conditions on K_p:

$$\begin{aligned} 0 < \quad K_p \quad &< \frac{1+a}{b} \\ 0 < \quad K_p \quad &< \frac{1+0.73}{45.18} \\ 0 < \quad K_p \quad &< 0.04 \end{aligned}$$

4.3 Experimental Results

In order to investigate the applicability of feedback control to autonomic replica management, the system described in Section 3 was simulated and the control loop depicted in Figure 5 and developed in Section 4.2 was implemented therein.

The system under study is a 15 node network maintaining 750 logical data items, with each node playing primary host to 50 distinct data items. Each access handled by the system refers to a single logical data item and may arrive at any node with redirects as necessary. Data access patterns are modelled using Zipf's Law. A Zipfian distribution models relative popularity of data where some items are accessed very frequently and others very rarely [1]. Placement of data items was such that in the absence of replication and in the presence of a steady arrival stream, all nodes would be equally loaded. Inter–arrival times are taken from an exponential distribution, thus creating a Poisson stream which are known to be a good approximation of independent arrivals from a large number of users [12, 13].

As a simple demonstration of the feedback control loop and its ability to maintain the desired response time, we compare system performance with feedback control (Figure 7) to performance experienced when no control is applied (Figure 6). In both cases the system runs through 20 sampling periods ($k \leq 20$) with a static stable configuration. Each sampling period is fixed at 15 minutes. This stable configuration consists of an inter arrival mean of 30ms, a traffic mix consisting of 50% read accesses and 50% write access, and 50% of data fully replicated. At the start of the 21^{st} sampling period, a disturbance is introduced; the arrival rate of write accesses is increased by 50% and remains at this new rate.

Due to the fact that write accesses are more expensive than read access and replication necessarily increases the cost of write accesses, a jump in average message response time is seen when the arrival rate is altered at $k = 20$. As expected, this jump in response time persists when no control mechanism is applied, Figure 6. In Figure 7 the feedback control loop alters (reduces) the percentage of data replicated and in so doing reduces the average cost of an update access, and restores average message response time to the desired value.

Figure 8 depicts the systems response to a temporary disturbance, an impulse. The system is in the same static stable configuration as described above for the first 20 sampling periods. At the start of the 21^{st} sampling period the arrival rate of write accesses is increased to 60% and reverts back to 50% thereafter.

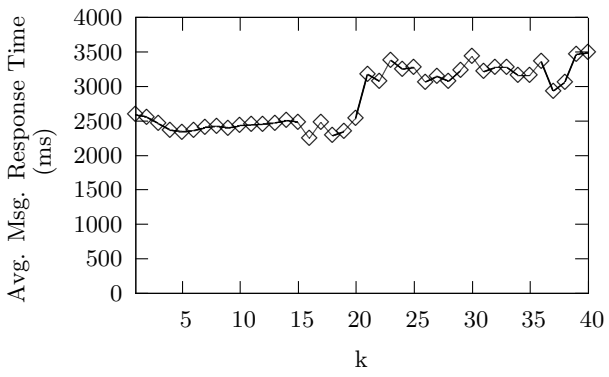

Figure 6: Without feedback control

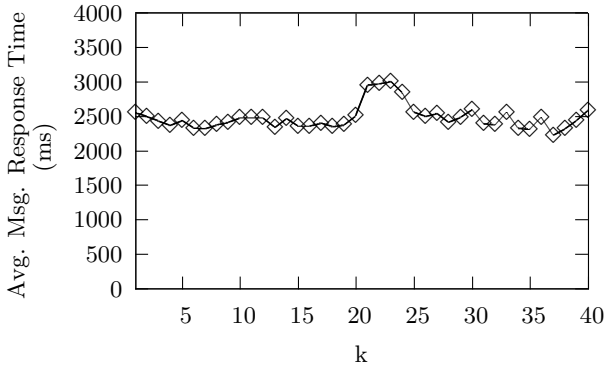

Figure 7: With feedback control

5. RELATED WORK

Though much research has been done in the area of distributed data management and replication, comparatively little considers update traffic [18] and fewer still consider a scenario where multiple distinct replication schemes are simultaneously applied to a single logical data item.

As a peer–to–peer system grows, so too do its resources, including bandwidth, storage space, and compute power. When combined with replication, this scalability and the inherently distributed environment yields a degree of failure transparency. Furthermore, when structured, a peer–to–peer network or Distributed Hash Table (DHT) not only offers a guarantee of an efficient route to every data item, but continues to do so in the face of changes in network topology. Though seemingly well suited to the milieu, sys-

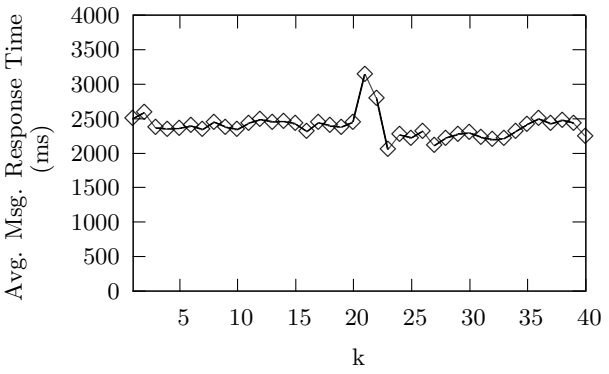

Figure 8: Impulse

tems built on top of DHTs (PAST [9], CFS [5]) are typically constrained by the decentralisation integral to peer–to–peer systems and do not maintain consistency. That is, data is read–only and is essentially cached as a means improving data availability and fault tolerance. As such, many of the more difficult issues relating to replication are ignored. Systems such as Ivy [16] accept updates but offer only relaxed consistency guarantees.

In direct contrast to the scalability and restrictive consistency guarantees of peer–to–peer systems, there exists a range of more centralised alternatives providing a wider range of consistency guarantees (Bayou [6], fluid replication [17], TACT [19]). The global information necessary for these systems, restricts scalability and applicability to a dynamic environment due to the possibility of frequent changes. Though these systems offer a multitude of consistency semantics across data–items, none offer a range of consistency guarantees for a single logical data item, Section 3.

6. CONCLUSIONS & FUTURE WORK

This paper has shown that autonomic control of replication schemes is possible, using a feedback control loop that monitors the mean response time of the system and adjusts the level of replication in order to maintain the response time close to a pre-determined reference value. In the simple scheme presented here, the replication management system is determined by a single controllable parameter, namely the percentage of data items that are fully replicated. A first-order difference model is sufficient to describe the relationship between the control input and the system output and a proportional controller succeeds in adjusting the replication scheme in response to changes in the output. In practice, there are far more facets to replication management. The following describes future work.

To date, optimisation through autonomic control in the distributed data layer has been predominantly data centric. How data is replicated and how consistency is maintained is altered to achieve desired performance metrics. Future work will also include feedback control of node parameterisations. Nodes in the system may not necessarily be dedicated, that is a node may have an alternate primary role in the system supported by the distributed data layer but additional resources allow said node to adopt a secondary role and contribute to both the data layer and the system it supports. In this respect, node resources could additionally be controlled using feedback control.

The selection of which data items are replicated has the potential to impact significantly on system performance, unused data–items selected for replication add nothing to performance and serve merely to consume storage. Conversely, a very heavily used data–item experiencing predominantly update traffic would degrade performance. Replica placement has the capacity to effect performance in a similar fashion. A data–item replicated in order relieve heavy load on one node that is placed on another heavily used node will offer little in terms of performance gains. As both replica selection and placement represent a high dimensional problem area (n data–items could be replicated to any permutation of m nodes), we are currently investigating the application of feedback control to the parameterisation of replica selection and placement algorithms. Feedback control is additionally used to determine the frequency with which a particular algorithm is run.

Application of feedback control to algorithm parameterisation and execution is being extended to apply to other aspects of replication management: including maintaining, response times, availability, resource usage and load balancing.

7. REFERENCES

[1] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 151–160, New York, NY, USA, 1998. ACM Press.

[2] M. Burgess and G. Canright. Scalability of Peer Configuration Management in Partially Reliable and ad hoc Networks. In *Proceedings of the* 7^{th} *IFIP/IEEE Conference on Network Management*, volume 246, pages 293–305, Colorado Springs, USA, March 2003. IFIP/IEEE, Kluwer.

[3] 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*, pages 1–17. IEEE Computer Society Press, 2002.

[4] 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*, pages 182–191. IEEE Computer Society Press, June 2004.

[5] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01': Proceedings of the Eighteenth ACM Symposium on Operating System Principles*, pages 202–215, New York, NY, USA, October 2001. ACM Press.

[6] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch. The bayou architecture: Support for data sharing among mobile

users. In *Proceedings IEEE Workshop on Mobile Computing Systems & Applications*, pages 2–7, Santa Cruz, California, August-September 1994.

[7] C. Doherty and N. Hurley. Policy–based autonomic replication for next generation network management systems. In *Proceedings 1st Annual Workshop on Distributed Autonomous Network Management Systems (DANMS)*, Dublin, Ireland, June 2006.

[8] C. J. Doherty and N. J. Hurley. Hierarchical policy–based replication. In *Proc. 26th IEEE International Performance, Computing and Communication Systems (IPCCC'07)*, New Orleans, Louisiana, USA, April 2007.

[9] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 75–80, Washington, DC, USA, 2001. IEEE Computer Society.

[10] 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*, 15(5):1266–1276, 2003.

[11] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[12] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, New York, NY, USA, 1 edition, April 1991.

[13] L. Kleinrock. *Queueing Systems*, volume Volume 1: Theory. Wiley-Interscience, 1975.

[14] 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*, pages 115–126, New York, NY, USA, 1997. ACM Press.

[15] 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*, 7(1), 1999.

[16] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. volume 36, pages 31–44, New York, NY, USA, 2002. ACM Press.

[17] B. Noble, B. Fleis, and M. Kim. A Case for Fluid Replication. In *Netstore '99: Network Storage Symposium*, Internet2, December 1999.

[18] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, pages 99–112, Berkeley, CA, USA, March 2004. USENIX.

[19] H. Yu. Tact: tunable availability and consistency tradeoffs for replicated internet services. *ACM SIGOPS Operating Systems Review*, 34(2):40, April 2000.

[20] M. Zach, D. Parker, L. Fallon, C. Unfried, M. P. de Leon, and J. N. Sven van der Meer, Nektarios Georgalas. Celtic initiative project madeira: A p2p approach to network management. In *Eurescom Summit 2005: Ubiquitous Services and Applications*, pages 27–29, Heidelberg, German, April 2005.