Self-Adaptation Applied to Peer-Set Maintenance in Chord via a Generic Autonomic Management Framework

Markus Tauber, Graham Kirby and Alan Dearle
School of Computer Science,
University of St Andrews, UK
Email: markus,graham,al@cs.st-andrews.ac.uk

***Abstract*—Self-adaptation can be achieved by *autonomic management* of facets of a system's constituent components. This paper reports on a generic autonomic management framework and on its application to a key-based routing protocol as used in the peer-to-peer overlay Chord. The framework implements generic components of the *autonomic management cycle*. In the work reported here it was used to build a manager which autonomically controls the maintenance scheduling of the *peer-set* in individual Chord nodes, governed by some high-level policies. This manager improved routing performance and resource consumption in comparison to statically configured Chord nodes in a deployed network which was exposed to various membership churn and workload patterns.**

***Keywords*-Self-adaptation; Autonomic Management Framework; Key-Based Routing; P2P Overlays**

I. Introduction

Self-adaptation is a category of behaviours exhibited by so-called *self-** systems which adapt their internal configuration parameters in response to a changing environment. An approach to achieve self-* behaviour is *autonomic management* as defined by Kephart and Chess in [1].

Autonomic management principles are based on a feedback loop (the *autonomic management cycle*) comprising four phases, namely:

- A *monitoring* phase, during which target-system-specific *events* are recorded.
- An *analysis* phase, during which *metrics* are extracted from the recorded events in order to represent the current situation.
- A *planning* phase, during which decisions are made about how to react to the current situation, driven by *policies*.
- An *execution* phase, during which the planned actions are carried out.

In this paper we introduce the *Generic Autonomic Management Framework (GAMF) [2]* which provides generic components of this autonomic management cycle and allows developers to focus on system-specific control logic.

We have used the GAMF to develop a manager that dynamically adapts the frequency at which the *peer-set* in Chord nodes [3] is maintained. As in any other P2P overlay that supports the key-based routing (*KBR*) abstraction [4], an individual Chord node maintains knowledge of the addresses of some subset of network nodes, its peer-set, this is similar to the *link-state* in protocols like *OSPF*. The peer-set is used for correct and efficient routing (referred to as *lookup*) and also to repair the network topology in the presence of membership churn. In the original Chord implementation [3] peer-sets are updated by periodic *maintenance operations*.

In a P2P/KBR overlay with a statically configured maintenance interval, the following unsatisfactory situations can be identified with respect to resource consumption and performance. The first arises when the frequency with which maintenance operations are executed is low and membership churn is high. The peer-set becomes inaccurate, leading to errors during the lookup process and to a reduction in performance. In this situation it is desirable to increase the maintenance frequency to increase peer-set accuracy. The converse situation arises when the frequency is high and churn is low (and/or no workload is executed). In this situation network resources are used unnecessarily, which may also reduce performance, making it desirable to decrease the frequency. We have hypothesised that such unsatisfactory situations will be identified and corrected via the autonomic management of the maintenance scheduling.

We have experimentally evaluated the effects of our autonomic manager on lookup performance and network usage in a deployed Chord network for various membership churn and workload patterns and compared them with a static configuration. In a majority of the experiments, significant improvements due to autonomic management were observed in the performance of routing operations and the quantity of data transmitted between network members.

This paper contributes to the area of networking research by presenting a self-* methodology which can be applied to peer-set maintenance in KBR protocols explicitly and which could be exploited for improving link-state maintenance. Additionally the introduction of the GAMF represents a contribution to autonomic management in its own right.

The reminder of this paper is as follows: In section II the architecture of the GAMF and its usage is explained. Following this we provide details of the autonomic manager and its implementation in Chord in section III. The experimental evaluation of the considered use case is outlined in section IV. In section V related work with respect to existing autonomic management frameworks and to peer-set related optimisation of P2P overlays is presented. This is followed by some concluding remarks and an outline of future work in section VI.

II. Framework Architecture

The GAMF is a *Java* framework based on an autonomic management cycle as illustrated in figure 1. The picture shows the four phases of the autonomic management cycle and corresponding GAMF-specific entities.

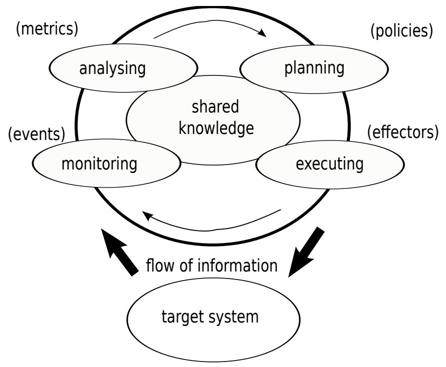


Figure 1: The autonomic management cycle [1] and corresponding entities (events, metrics, policies, effectors).

The generic components of the autonomic management cycle that are provided by the GAMF are:

- A monitoring facility, for triggering event generation
- A monitoring data store (*shared knowledge database*), for:
  - thread-safe provisioning of monitoring data
  - basic filtering for event types and age
- A scheduling facility for triggering of the:
  - extraction of metrics
  - evaluation of policies
- A register for meta-data

Interfaces are provided for these generic components in order that system-specific management components (*system adapters*) may interact with the GAMF. The operations of these system adapters correspond to individual management cycle phases.

A set of system adapters for a particular target system may be composed with the GAMF to yield an autonomic manager for that system. In turn, the autonomic manager may be composed with the target system to yield an autonomic system.

The system adapters include *event generators* and *effectors*, which allow interaction of the control mechanism with the target system. They also include *metric extractors* and *policy evaluators*, which provide the means for computing a specific response, determined by policies, to an observed situation, modelled by metrics. In more detail:

- *Event generators* provide the GAMF with time-stamped information about specific *events* in the target system. An event comprises an event type, a time-stamp and a field for additional information, specified by the system adapter developer. In order to allow unambiguous usage of event types, the types of events generated by an individual event generator can be registered with GAMF to prevent them being used by other event generators.
- *Metric extractors* are used to extract monitoring data from the shared knowledge database in order to represent a specific situation, modelled by the *metric*. The metric is specified by the system adapter developer with a metric type, a time-stamp specifying the computation time of the metric value, a field a numerical metric value, and another for additional information.
- *Policy evaluators* evaluate the policy specified by the system adapter programmer. A policy determines the action to be carried out in response to the target system's current situation (represented by specific metric values).
- *Effectors* carry out specific actions in the target system. An effector may be triggered by a policy evaluator in order to change a controlled system configuration parameter and to make the system aware of the change.

The GAMF includes a flexible mechanism to filter for specific events or metrics in the shared knowledge database, allowing filtering by type and by the time of recording.

Metric extractions and policy evaluations may be:

- scheduled at regular intervals;
- triggered by the arrival of a specific event type; or
- triggered on an arbitrary schedule.

III. Autonomic Management applied to Chord

Our manager is intended to detect when maintenance effort is being wasted and to decrease the current maintenance rate accordingly. Conversely, it increases the rate in situations when more vigorous maintenance is appropriate.

The potentially conflicting goals of reducing effort and increasing performance are each individually managed by a sub-policy. During the planning phase, each sub-policy makes its own independent recommendation as to how the current maintenance rate should be adjusted. The mean value of these recommendations is then applied during the execution phase.

The rationale for this structure is that although the sub-policies will rarely agree, their recommendations will cancel out in situations where little action is required, whereas in more extreme situations one will outweigh the other, due to the magnitude of the recommendations.

During each planning phase, each sub-policy considers metric values derived from events received during the current autonomic cycle. These events are based solely on locally gathered data, thus no additional network traffic is generated by the autonomic manager.

A. Monitoring

An event is generated whenever:

- a maintenance operation is executed without any effect on the peer-set

- a failed attempt to access a peer-set element is made, either during routing or during a maintenance operation

B. Analysis

Two metrics aggregate the events. The *Wasted Maintenance Count* (WMC) and *Error Count* (EC) metrics count the numbers of each event type during the current autonomic cycle.

WMC models the amount of effort invested in maintenance operations without effect. A high value suggests that a lot of network traffic was unnecessary, since either little churn occurred, or maintenance was executed frequently enough to compensate for such changes. Conversely, a small value implies that the network traffic due to maintenance operations was effective in correcting errors, and may therefore be regarded as justifiable.

EC models the accuracy of the peer-set as perceived by user or maintenance activities attempting to use it. A high value suggests that a large proportion of the peer-set is not valid. Conversely, a low value suggests one of: a high degree of accuracy in the peer-set due to frequent maintenance; low churn; or low user demand due to a light workload.

C. Planning

Each of the sub-policies is driven by one of the metrics. The sub-policy concerned with reducing effort considers WMC, while the sub-policy concerned with improving performance considers EC. The rationale for the former is straightforward; for the latter, the point is that user-level routing operations will retry when errors are encountered, thus a high error rate leads directly to poorer performance. As with any negative feedback approach, each sub-policy recommends a change to the current maintenance rate, of a magnitude related to the difference between the current value of the relevant metric and some ideal value for that metric. The further that the metric diverges from the ideal, the more aggressive the response that is recommended. Since both metrics count undesirable events, the ideal value for both metrics is zero. Whenever WMC is non-zero the sub-policy concerned with reducing effort recommends a reduction in the maintenance rate, while whenever EC is non-zero the sub-policy concerned with improving performance recommends an increase in the maintenance rate. For ease of integration with Chord, in practice each sub-policy recommends a new maintenance interval, rather than a rate. It calculates the proportion P by which the current interval should be changed. The new interval is then calculated as:

$$new\ interval = current\ interval \times (1 \pm P) \qquad (1)$$

Where the sub-policy using WMC uses $(1+P)$ and EC in contrast uses $(1-P)$ as it seeks to increase the maintenance rate. In both cases, the proportion of change P lies between zero and one, and is calculated as:

$$P = 1 - \frac{1}{\frac{metric - ideal}{k} + 1} \qquad (2)$$

where $metric$ denotes either WMC or EC as appropriate. $ideal$ is zero in both cases. k is a dampening factor for each sub-policy, a positive constant that controls the rate of change of P with respect to the difference between the metric value and its ideal value. The higher the value of k, the lower the resulting proportion of change, and hence the slower the resulting response by the manager. Figure 2, shows the proportion of change resulting from various EC metric values, for a range of k values. The overall response

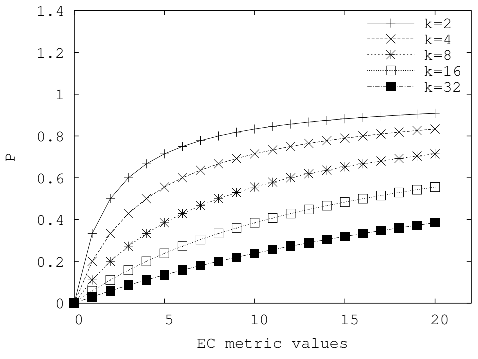


Figure 2: Relationship between EC and P for various k

of the policy is to set the maintenance interval to the mean of the values recommended by the sub-policies. In addition, if any error has been detected during the current cycle, a maintenance operation is invoked immediately. This is intended to improve reaction to phase changes when the error rate increases rapidly. Without this action, errors occurring during a long maintenance interval (set due to a currently low error rate) would not be rectified until the end of the interval.[1]

D. Execution

Once the planning phase is complete, the execution phase involves invoking a maintenance operation if necessary, and setting the new value for the Chord node's maintenance interval.

E. Implementation

In Chord, peers are identified via a key drawn from a circular key space. The peer set of each node is comprised of the node's *successor(s), predecessor* in key space and *fingers* which point across the key space. We have used an in-house *Java* implementation of the original Chord P2P overlay[2] which maintains each of these independently. The manager

[1]Note that the length of a maintenance interval may be considerably longer than the duration of an autonomic cycle.

[2]http://www-systems.cs.st-andrews.ac.uk/wiki/Software

described here controls all three of the maintenance operations separately. This was achieved by adding corresponding `setMaintenanceInterval()` interfaces to be used as effectors. An event generator was also added to Chord to trigger the recording of the desired monitoring information (see III-A). The above code executes in the same address space as a Chord node, asynchronously from the operations of an unmanaged node.

IV. Experimental Evaluation

A. Overview

The effects of the autonomic manager on Chord's performance and network usage were measured in a sequence of experiments. Each experiment combined a particular:

- workload (a temporal pattern of lookup requests),
- membership churn (a temporal pattern of nodes joining and leaving the overlay), and
- scheduling policy.

For each experiment, a fixed number of Chord nodes were deployed in an isolated test-bed of 16 machines. Each node was deployed on a separate physical machine to avoid competition for CPU and network resources on individual machines. Each experiment was repeated three times.

B. Workloads

- A *synthetic light-weight workload* represented scenarios in which few lookups were executed. A total of 10 lookups were issued, with 300 seconds of inactivity between each one.
- A *synthetic heavy-weight workload* represented scenarios in which lookups were executed at a high rate. A total of 6,000 lookups were issued, with no delay between each one.
- A *synthetic variable-weight workload* represented scenarios involving temporal variation in workload intensity. A total of 1,000 lookups were issued, in batches of 100 successive lookups followed by 300 seconds of inactivity.
- A *file system workload* simulated a Chord workload derived from a real world file system trace [5] applied to a distributed file system built on Chord.[3] The resulting workload contained 15,000 lookups.

Whereas the synthetic workloads were all sequential, the file system workload contained a mix of sequential and parallel lookups, since the lookups required for some file system operations (such as locating all replicas of a file) can be performed in parallel.

[3]This research took place within the context of work using a P2P overlay as a platform for a distributed file system.

C. Churn Patterns

Each churn pattern modeled the behaviour of a set of nodes, in terms of a sequence of alternating on-line and off-line phases for each node.

The first two churn patterns represented uniform behaviour among all nodes, one pattern with a low churn rate and the other with a high churn rate. The durations of the on-line and off-line phases were pseudo-randomly generated from specified normal distributions.

Two other churn patterns represented networks exhibiting both low and high churn rates. In one, the behaviour of any given node exhibited either low or high churn consistently throughout the experiment. The final churn pattern involved a series of phases during which the whole network switched repeatedly between low and high churn rates.

- In the *low churn pattern* nodes only exhibited a short initial period (up to three minutes) of churn in which nodes joined and formed the overlay. Once the overlay was fully established no node left the overlay again.
- In the *high churn pattern* all nodes exhibited a sequence of alternating on-line and off-line phases with durations drawn from the following normal distributions:
  - on-line: $\mu = 200s, \sigma = 40s$
  - off-line: $\mu = 100s, \sigma = 20s$
- In the *locally varying churn pattern* 25% of the nodes exhibited the low churn behaviour described above, and 75% of the nodes exhibited high churn behaviour.
- In the *temporally varying churn pattern* the entire network exhibited alternating phases (each $\approx 1,000$s) of low churn and high churn behaviour.

D. Scheduling Policies

The following scheduling policies were used:

- A null policy, *policy 0*, made no dynamic changes to the maintenance interval. For fair comparison, this was implemented using the same mechanisms as the autonomic policies.
- A 'relaxed' autonomic policy, *policy 1*, used high dampening factors (8 for WMC and 32 for EC), yielding a relatively slow response to unsatisfactory situations.
- An 'aggressive' autonomic policy, *policy 2*, used low dampening factors (1 for both WMC and EC), yielding a relatively rapid response to unsatisfactory situations.

In all cases the duration of the autonomic cycle was set at 2s, thus the policies were evaluated every 2s. The initial default maintenance interval was also set at 2s. For both policy 0 and policy 1, the fixed static interval and dampening factors were derived from preliminary experiments in a subset of the above conditions (see [6]).

E. Evaluation Criteria

The effectiveness of the various policies was evaluated in terms of impact on Chord performance as perceived by the user, and on network traffic generated by Chord. Network usage was measured simply as the mean outgoing data rate for all nodes.

The chosen performance metric was *expected lookup time*, defined as the mean overall duration of lookup operations, under the assumption that the caller retries repeatedly on error until a result is obtained. A practical motivation for this simplified approach was to combine the performance-related measurements for successful and failed lookups as listed below:

- *lookup time* t_{lookup}, the mean duration of individual successful lookup operations
- *lookup error time* t_{error}, the mean duration of individual failed lookup operations
- *lookup error rate* p_{error}, the probability for a lookup operation to fail

The *expected lookup time* comprised the cost of the eventual successful lookup plus the weighted sum of all possible sequences of successive failures:

$$t_{lookup} + \sum_{i=1}^{\infty} i \times t_{error} \times p_{error}^{i} \tag{3}$$

The *expected lookup time* (ELT) was calculated for each metric for each successive 5 minute time-window during the course of each experiment. The network usage (NU) was also extracted for every 5 minute interval (no post processing was necessary). This allowed us to plot the metric values over time and use them for summarising and quantifying the effects. Alternative evaluation methods are described in [6], [7].

F. Results

1) Overview: The experiments comprised all combinations of the four workloads and four churn patterns. Table I shows the number of experiments in which each policy yielded the best results, for ELT, NU, and for both together.

	ELT	NU	both
policy 0 (null)	2	1	0
policy 1 (relaxed)	8	3	0
policy 2 (aggressive)	6	12	5

Table I: Number of experiments 'won' by each policy

Table II provides an holistic view of the effects of autonomic management. It shows the mean ELT and NU values in managed systems normalised to an unmanaged system. Thus every normalised value less than 1 represents a benefit of the specific autonomic management policy with respect to the unmanaged system.

		policy 1		**policy 2**	
workload	*churn*	ELT	NU	ELT	NU
light-weight	low	0.725	0.09	0.705	0.028
	high	**0.814**	0.548	**0.817**	0.353
	local	**0.835**	0.454	**1.207**	0.438
	temporal	0.807	0.293	**0.979**	0.178
heavy-weight	low	0.727	0.314	0.698	0.23
	high	0.605	**1.333**	0.693	**0.983**
	local	0.085	**1.267**[4]	0.183	**1.082**
	temporal	0.562	0.541	0.672	0.4
variable	low	0.714	0.111	0.7	0.054
	high	0.362	**1.202**	0.364	**0.781**
	local	**3.239**	0.416	2.954	0.421
	temporal	**1.559**	0.258	**0.974**	0.245
file-system	low	0.804	0.341	0.787	0.293
	high	5.142	0.47	**1.089**	0.523
	local	0.6	0.453	0.592	0.882
	temporal	0.862	0.595	**0.932**	0.409

Table II: Normalized performance and network usage

Bold and underlined values in table II represent results which were not statistically significantly different from the baseline (policy 0), according to a *visual approximation test for significance* under the consideration of a *90% confidence interval of the mean* (ci_{μ}). The comparison of expected lookup times abstracts over the variation of the underlying raw data (*lookup time*, *lookup error time* and *lookup error rate*) for simplicity. Samples of the raw data as provided in [6] for each measurement show that statistically significant differences can be identified in the majority of the raw data sets used to derive the expected lookup time. As an example figure 3 shows the ci_{μ} (with 90% confidence) for *lookup times* in experiments with heavy weight workload and high churn comprising data sets of an average size of $\approx 15,000$.

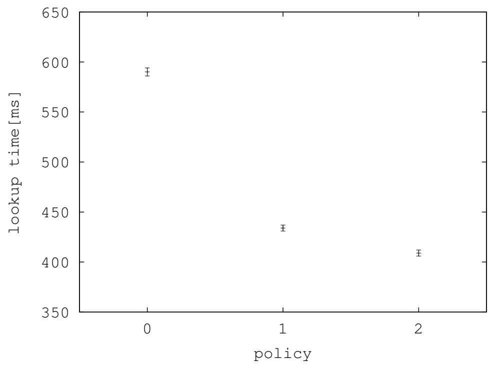
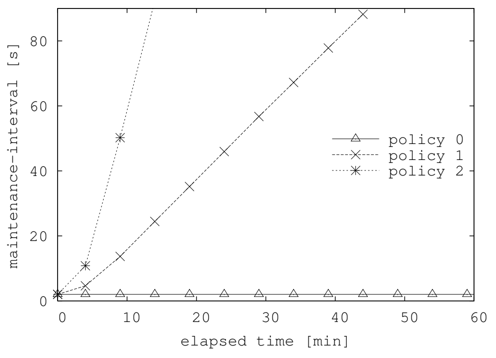
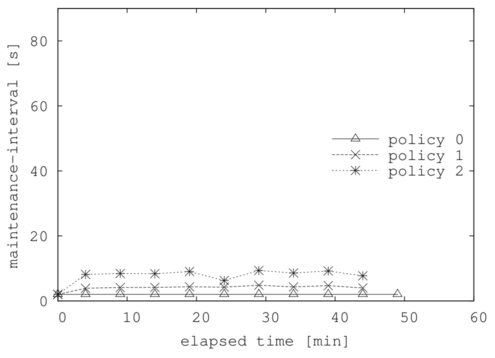


Figure 3: Lookup Times (raw data)

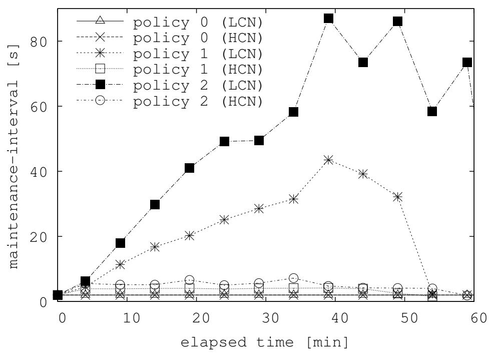
[4]In contrast to the visual approximation, a t-test results in a significant difference in this case.



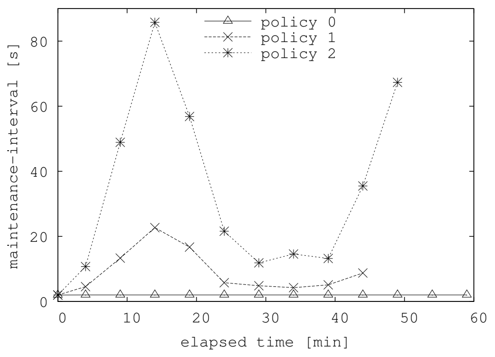
(a) Low churn



(b) High churn



(c) Locally varying churn



(d) Temporally varying churn

Figure 4: Interval progressions for heavy-weight workload and various churn patterns

2) Autonomic Manager Behaviour: To illustrate the policy actions resulting in the effects reported in table II, we plot the progression of maintenance intervals over time (we omit error bars for readability). Figures 4a-4d show the progressions of the maintenance intervals over the courses of the experiments, for the four churn patterns. Each point plotted is the mean of the corresponding figures for three repeated runs. In the experiment with locally varying churn, the progressions are plotted separately for low-churn nodes (LCN) and high-churn nodes (HCN).

Figure 4a shows that for low churn, the autonomic policies detected an unsatisfactory situation with respect to *network usage*, and reacted by steadily increasing the maintenance interval. This decreased the amount of work each node spent (unnecessarily) maintaining its peer-set, and thus reduced the amount of data sent to the network in comparison with unmanaged nodes. Additionally, a reduction in the work spent on maintenance operations left more computational capacity for dealing with lookup operations. This reduced the *expected lookup time*. As expected, *policy 2* increased the interval more aggressively than *policy 1*.

Figure 4b shows that for high churn, the autonomic policies held the intervals fairly constant, though at higher values than for unmanaged nodes. Referring to table II, this yielded roughly the same *network usage* as for unmanaged nodes, and a significant improvement in *expected lookup time*. This improvement in performance may appear counter-intuitive, given the reduction in overall maintenance effort, particularly since examination of the experimental logs shows that error rates were significantly *lower* for the autonomic policies than for unmanaged nodes. The explanation is that each value plotted is derived by averaging individual maintenance interval values over the entire network, and over a five minute aggregation time window. This masks the fact that there was considerable variation in controlled interval values within each time window. Whenever the manager of a given node detected errors in its peer-set, it immediately decreased the maintenance interval, giving a period of high maintenance activity. Once the errors were corrected, the manager increased the interval again until the next error. Thus errors were corrected more rapidly than in an unmanaged system, despite the overall average interval being higher. Again, *policy 2* reacted more aggressively than *policy 1*, and kept the maintenance interval at higher levels.

Fig. 4c shows the resulting intervals for locally varying churn, where some nodes (75%) exhibited high churn, and the rest, low churn. There are two features of interest: the apparent phase change after about 40 minutes, and the fact that the autonomic managers behaved markedly differently on the low churn and high churn nodes.

We have no simple explanation for the phase change, other than to hypothesize that the particular churn patterns in use caused some threshold in the error rate to be exceeded, triggering rapid decreases in the intervals.

The differences in behaviour between low and high churn nodes appear anomalous, since all nodes experience roughly the same environment in terms of the aggregate behaviour of their peers (assuming that the low churn nodes are uniformly distributed throughout the network). The explanation is that a node's maintenance interval was reset to the default value every time it restarted, in order to simulate the arrival of new nodes in a network. Thus each manager on a high churn node did react to the environment in the same way as the low churn nodes, by steadily increasing the maintenance interval, but since the interval was regularly reset, the average value was held fairly constant and close to the default value.

Figure 4d shows the autonomic policy behaviour for temporally varying churn, in which the entire network alternated between low and high churn, in phases lasting about 17 minutes. During the initial low churn phase the managers responded as expected, in the same way as in the low churn experiment. When the network moved into high churn they reacted by decreasing the maintenance intervals. The more aggressive behaviour of *policy 2* can be seen clearly. Table II shows that this gave slightly better results for *network usage* than *policy 1*, but slightly worse for *expected lookup time*. Both policies obtained significantly better results than the unmanaged network.

V. Related work

A. Autonomic Management Frameworks

The motivation for developing the GAMF was that existing tools such as *vGrid* [8], *AutoMate* [9], *k-component* [10], *IBM autonomic computing toolkit* [11], [12] and *Accord* [13] were considered too complex and heavy-weight for the context in which this work was carried out.

AutoMate and Accord are expected to have a significant impact on the target system's runtime performance as they contain P2P overlay networks for discovering components in a distributed system, and mechanisms to extract policies from *XML* formatted configuration files.

The *IBM autonomic computing toolkit* allows the application of autonomic management using a wide variety of built-in *interfaces* which are however limited in their scope. These interfaces require the manager to operate in a different address space to the target system. This and the complex machinery which comes with this tool impose a significant load on the target system.

For similar reasons, the *k-component* architecture was not considered in this work. It provides a *C++* library which defines an *Adaptation Contract Description Language (ACDL)* to specify how a controlled component is adapted by some control mechanism. A *Collaborative Reinforcement Learning* methodology is used to gather information from remote components via *CORBA*. Component meta-information, used for the ACDL, is stored in separate files in XML format. This meta-information is generated via the use of addtional tools.

The advantage of the GAMF over all of the above is that it i) has a simple interface, ii) is thus very flexible, iii) and is capable of running in the same address space as the target system.

B. Key-Based Routing Optimisations

The most closely related work is based on Pastry [14] and describes the optimisation of resource consumption for a given acceptable message loss rate. This is a form of performance metric. Each node dynamically estimates the overall node failure rate and the size of the overlay, and uses an analytical model to deduce an appropriate maintenance rate that should yield the target loss rate. Our approach is simpler in that it does not require estimation of any global properties, instead applying simple heuristics based on local observations.

In contrast to our approach, [14] does not take user workload into account. Furthermore, it imposes a lower bound on resource consumption, since the maintenance rate is not further reduced once the acceptable loss rate is achieved, even in situations where resource consumption could be lowered while still meeting the target loss rate.

Binzenhöfer and Leibnitz [15] describe how churn may be estimated in Chord networks in order to set maintenance rates appropriately. They do not however perform any adaptation. Churn is estimated by monitoring changes in a node's peer-set. The focus is on limiting the probability of network partition before the next maintenance operation. This approach does not take user workload into account; the churn estimator receives only information gathered during maintenance operations. Consequently, the algorithm may be slow to react to a sudden increase in churn occurring during a low-churn period with low maintenance rates.

Chord2 [16] aims to reduce maintenance costs by introducing a two-level structure, with a smaller ring of high performance super-peers used to manage finger tables. There is no dynamic control of maintenance intervals.

VI. Conclusions

We have demonstrated that autonomic management of maintenance scheduling in Chord can achieve significant improvement in performance and resource consumption.

Under changing conditions, autonomic management can adapt scheduling to suit prevailing conditions. Under static conditions, it can converge to a better scheduling than is likely to be configured for an unmanaged system.

There are several avenues for possible further development of our management approach. Some involve refinements to Chord itself (for instance a fall-back mechanism to compensate for accesses to faulty finger table entries), while others involve different approaches to autonomic management (for instance cascading managers which adapt the dampening factor autonomically). Our autonomic management approach could also be applied straightforwardly to other P2P overlay networks that perform periodic maintenance operations, at statically configured intervals, such as Tapestry [17], CAN [18] and Pastry [19].

This approach could also be exploited to build managers for adapting peer-set or link-state maintenance scheduling in network protocols outwith the P2P research area.

The GAMF has also been used to control the degree of concurrency in data retrieval operations in a distributed storage system [6], supporting the claim for GAMF's genericity and flexibility. The sub-policy based approach to autonomic management as reported here is, together with the GAMF, applicable to a wide range of topics and we hope that this motivates its usage outwith the scope of this work.

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] M. Tauber, "A Generic Autonomic Management Framework (GAMF)," February 2010. [Online]. Available: http://www-systems.cs.st-andrews.ac.uk/gamf

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," in *ACM SIGCOMM 2001*, August 2001, pp. 149–160.

[4] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," in *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[5] D. Roselli, "Characteristics of File System Workloads," University of California at Berkeley, Tech. Rep. CSD-98-1029, 1998. [Online]. Available: citeseer.ist.psu.edu/736324.html

[6] M. Tauber, "Autonomic Management in a Distributed Storage System," Ph.D. dissertation, University of St Andrews, School of Computer Science, 2010, arXiv:1007.0328v1.

[7] M. Tauber, G. Kirby, and A. Dearle, "Autonomic Management of Maintenance Scheduling in Chord," University of St Andrews, School of Computer Science, Tech. Rep. arXiv:1006.1578v1, 2010.

[8] B. Khargharia, S. Hariri, M. Parshar, L. Ntaimo, and B. Kim, "vGrid: A Framework For Building Autonomic Applications," in *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Enviroments (CLADE'03)*, 2003.

[9] M. Agarwal, V. Vhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, and M. Parashar, "Automate: Enabling autonomic applications on the grid," Department of Electrical and Computer Engineering, Rutgers University, Seattle, WA, CAIP TR-269, 2003.

[10] J. Dowling and V. Cahill, "The K-Component Architecture Meta-Model for Self-Adaptive Software," in *Proceedings of 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (Reflection2001)*. Springer-Verlag, 2001, pp. 81–88.

[11] N. Chase, *Understand the Autonomic Management Engine*, IBM, June 2004. [Online]. Available: https://www6.software.ibm.com/developerworks/education/ac-ame/

[12] B. Melcher and B. Mitchell, "Towards an Autonomic Framework: Self-Configuring Network Services and Developing Autonomic Applications," *Intel Techology Journal*, vol. 8, no. 4, pp. 279–290, November 2004.

[13] H. Liu and M. Parashar, "A component based programming framework for autonomic applications," in *the International Conference on Autonomic Computing*, New York, NY, USA, 2004.

[14] R. Mahajan, M. Castro, and A. I. T. Rowstron, "Controlling the Cost of Reliability in Peer-to-peer Overlays," in *2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*. Springer, 2003, pp. 21–32.

[15] A. Binzenhöfer and K. Leibnitz, "Estimating Churn in Structured P2P Networks," University of Würzburg, Tech. Rep. 404, 2007.

[16] J. Yuh-Jzer and W. Jiaw-Chang, "Chord2: A Two-Layer Chord for Reducing Maintenance Overhead via Heterogeneity," *Computer Networks*, vol. 51, no. 3, pp. 712–731, 2007.

[17] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, January 2004.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," in *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2001, pp. 161–172.

[19] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer ystems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.