Probabilistic Modelling of Replica Divergence

Antony I. T. Rowstron, Neil Lawrence and Christopher M. Bishop
Microsoft Research Ltd.
St. George House, 1 Guildhall Street,
Cambridge, CB2 3NH, UK.
antr@microsoft.com

Abstract

It is common in distributed systems to replicate data. In many cases this data evolves in a consistent fashion, and this evolution can be modelled. A probabilistic model *of the evolution allows us to estimate the divergence of the replicas and can be used by the application to alter its behaviour, for example to control synchronisation times, to determine the propagation of writes, and to convey to the user information about how much the data may have evolved.*

In this paper, we describe how the evolution of the data may be modelled and outline how the probabilistic model may be utilised in various applications, concentrating on a news database example.

1. Introduction

In distributed systems the replication of shared mutable data has been widely studied. When mutable data is replicated there is a need to consider the consistency model used to control the level of divergence of the different replicas.

In this paper, we advocate using knowledge of how the shared data evolves to control and manage divergence. Empirical evidence shows that updates to shared data, in many cases, follow systematic patterns. By modelling the way in which the data has been updated in the past, we can provide information to an application on how, the data has evolved since the replicas were last consistent. The basis of this approach is *probabilistic modelling* applied to the distribution of operations performed on the data structure. The approach is novel and preliminary results on a mobile news database and a mobile email reader are encouraging.

In the next section we describe the general approach, in Section 3 a mobile news database case study is detailed, in Section 4 the results for a mobile email reader are presented and then in Section 5 we describe other applications we are currently working on.

2. The General Approach

An application using our approach is provided with probabilistic models that capture how a replicated data structure evolves in time. These probabilistic models allow the application to estimate the number of updates that are likely to have been performed on the data structure, or part of it, during a specified time period, for example between the last time a synchronisation was performed and the current time. The application can then use this to adapt to the data structures' evolution by, for example, controlling when synchronisations should occur, alerting the user to divergence, or controlling when updates to the shared data are propagated. The generation of a single probabilistic model that captures the evolution of the other replicas, is known as *inference*. The application then makes *decisions* based upon the information contained within this single model. This partition of the problem into two stages of inference and decision enables our approach to be applied to a wide variety of applications. The inference stage is decomposed into the generation of models representing the evolution of the replicated data structure, or parts of it, and the subsequent combining of these models as requested by the application. In the For the inference stage a general purpose tool can be used to create the probabilistic models, and combine them, whilst the decision stage is specific to each application.

The probabilistic models are generated by a tool, which requires a log of descriptions of the operations performed on the shared data structure. For each update to the data structure the log contains: information about the operation, the part of the data structure that was affected, the time when the operation was performed and an identifier representing the source of the update (for example a user id). A description of the data structure and its different components is also required by the tool, which allows each component of the data structure to be modelled independently. Once a set of probabilistic models have been created, these can be updated dynamically as new updates are performed.

As a simple example of the data structure decomposition,

consider an address database application. Each entry within the database is marked as either being a personal contact or as a business contact. This data structure can be thought of as being composed of two parts, the personal and the business parts, and two models can be generated and provided to the application. A model for the entire address database can then be generated by combining the two models. A further sub-division could exist within the database with, perhaps, the personal database is divided into family and friends. Separate probabilistic models can then also be generated for each of these sub-divisions and again composed.

The application is required to create the logs, provide the information about the decomposition of the data structure, and to perform the decisions based upon the information provided by the models.

Probabilistic Modelling Learning of the probabilistic models can be automated using model selection techniques, and the models may also be changed over time as more updates are made to the replicas.

A probabilistic model is a particularly powerful representation as such models may be combined in a straightforward and principled manner. This means that the data structure can be decomposed and each part modelled individually, as can different sources of data updates. For example, in the address database application, there could be separate models for the secretary and the owner of the address book, reflecting their particular patterns of updates. Hence, when the address book is replicated for the owners use, the probabilistic model generated can describe how the secretarys copy evolves.

It is important to remember that the probabilistic model is a prediction of future behaviour based on the past. The true evolution of the replica may not be consistent with the model. Even if the model is correct, its probabilistic nature means that its individual predictions can err, even if in general it is accurate. As a result we advocate using our approach in ways that will enhance the user experience rather than restrict functionality. The user interface should suggest and advise rather than constrain and command.

The System The System is composed of a tool for creating the probabilistic models, and a library for use in the application for merging the probabilistic models. These models capture the rate at which the operations are performed, and how that rate changes over time. Therefore, the time at which an update to the data structure occurs is the primary information required to create the models. The other information in the log allows multiple models to be created, based on the part of the data structure being updated, or on the user performing the update. In order to achieve this, the tool pre-processes the log, creating a separate log for each entity to be modelled. A probabilistic model is then created for each of these sub-logs independently, and this is now described.

There are a number of factors that effect the creation of the models. For example, the periodicity of the data has to be determined (e.g. hourly, daily, weekly, monthly and so forth). The tool currently creates histogram based models. Such models may be parameterised by widths and starting points for the bins. All the parameters of the model can be learned from the information contained within the log. It should be noted that there are many alternative forms of probabilistic model which can be used, for example wrapped mixtures of Gaussians and circular normals (see [5]). Although in this paper we use histogram based models, we are currently evaluating other approaches.

For each probabilistic model the correct parameters need to be established, and these control the model complexity. The main consideration in the selection of the model complexity is its generalisation ability. In other words, we wish to create a model that not only describes well the updates upon which it is based but also one that will describe future updates. In the address book example above, where the events we are modelling are updates of the database, we could create a histogram model with too many bins so that each update occurs in a single bin. Such a model is unlikely to provide a good predictor of the evolution of the replica because the model complexity is too high. At the other extreme, if we create a histogram model with only one bin we will be predicting a uniform distribution for the future updates, again this is likely to be a poor predictor of the replica's evolution. There is obviously a 'happy medium' and this may be found through *cross-validation* of the model [1]. Cross-validation involves splitting the log into a number parts, for example five. The first four parts are then used to construct a model with a particular parameterisation and the fifth part is used to 'validate' the model. This involves computation of the histogram models likelihood of creating the validating data. The part that is used for validation and one of those used for construction is then inter-changed and the model is re-validated. This procedure is repeated five times so that each part of the data has been used to validate the model once giving five different scores. The validation scores are then combined, for example by averaging, and the final score is associated with the parameters used for constructing the model. A range of parameterisations can be tested in this manner and the one with the highest score is then selected, and utilised to construct a model based on all the data, which is the final model.

Another factor determined during the cross-validation phase is the periodicity of the updates. The tool uses a number of pre-programmed periodicities: a daily cycle, a weekly cycle, weekdays separately generated from weekends, and Saturdays separately generated from Sundays both of which are separately generated from weekends.

More pre-programmed periodicities can easily be added, such as hourly or monthly based periodicities. Note that the set of candidate models includes the uniform distribution, and so the performance of the system should be no worse that that of the uniform model, in the event that one of the pre-programmed periodicities is not appropriate. Currently, we are looking at other techniques to determine the periodicity of the updates.

A *prior distribution* is used, which can either be a *uniform prior* or, in some situations, there may be prior knowledge about when the updates arrive. The prior distribution is combined with the model generated using Bayes's rule. For a histogram model this prior plays an important role of ensuring the final histogram model is non-zero at all points within its range, i.e. even when there are no observed points within a bin's range the histogram has some residual value. The residual value of the histogram is a further cross-validated parameter. If there is no or little information about when the updates occur, this is valuable, because the model is initialised using the prior distribution and as more updates are observed, the model is refined to represent more accurately the underlying distribution of the updates. This process can be extended to allow more recent data to be more influential, thereby allowing the system to deal with non-stationary situations in which the distribution is itself evolving with time.

3. Example Mobile News Database

We now demonstrate the use of our approach in a mobile news database application. We are seeing a proliferation of applications that replicate information on mobile devices, such as the service provided by AvantGo[1]. These allow mobile devices to store replicas of small news databases for access when the mobile device is disconnected.

Our mobile news database application provides, on a mobile device, a list and summary of the current news stories. We assume that the mobile device has wireless connectivity which allows it to synchronise with the master news database. We assume a pull model, where the device initiates the connection.

For this application a database of new articles is required, and we generated one from the BBC News web site. The BBC publishes news articles to their web site 24 hours a day and each of the news items is classified under a category, such as sport, business, health and so forth. For every article appearing on the BBC News website over a three month period we extracted the date and time of publication, and the subject of the news article to create a news database.

We treated the news database as the replicated data structure, and used the information gathered from the website to create the log required to generate the probabilistic models. We decomposed the news database into several parts, where each subject was treated as a separate part. All writes to the news database were considered as being performed by a single user. The mobile news database application allowed the user to select which parts of the news database they wished to have replicated on the mobile device.

The probabilistic models of the news database are created by the tool overviewed in the previous section. The mobile news database uses the probabilistic models to generate a visual cue in the application interface to allow a user to see the number of updates that are likely to have occurred to each part of the news database since the last synchronisation. The application also uses the probabilistic models to control synchronisation times between the device and the master news database. It is likely that, due to the cost of bandwidth, as well as the limited amount of bandwidth, the mobile devices will not be continuously connected. Therefore, the synchronisation times have to be chosen, as this is part of the decision stage.

Optimal synchronisation The obvious approach to choosing when the mobile device should synchronise would be to have a user specify the number of synchronisations per day they were willing to pay for[2], and these would occur uniformly during the day, for example once every four hours.

Our mobile news database makes an adaptive choice of when to synchronise. This aims to find a trade-off between the cost of synchronisation and the average staleness of the data, where staleness is defined as the time between an article appearing on the master news database and appearing on the device.

In order to calculate the synchronisation times, it is necessary to formalise the user's requirements and calculate how to achieve them. In the mobile news database this is achieved either by setting the number of synchronisations per day to achieve a particular average staleness, or by allowing the user to set the number of synchronisations per day and then scheduling the synchronisation times to minimise the staleness.

We express the user's preferences in terms of a *cost function*, which represents mathematically what the user wants. In the news database problem, the simple cost function we have chosen is one which represents the staleness of the news articles. We wish to minimise the time that articles are available in the news database, but are not available on the mobile device. For every article the staleness is the time from when the article was available but not in the replica on the mobile device.

[1] http://www.avantgo.com/

[2] Assuming a per synchronisation charge; other charging models are possible and different cost functions can be created to deal with this in our approach.

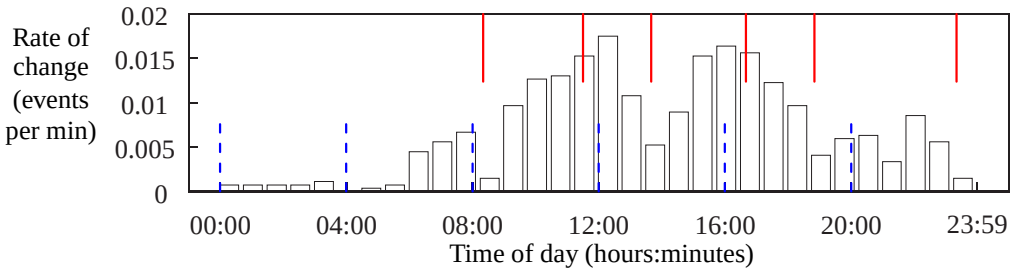

Figure 1. Synchronisation time optimisation for the Business part of the news database, showing the learned histogram model together with uniform synchronisation times (dashed lines) and the optimised synchronisation times (solid lines).

The cost incurred from synchronising at time s_i may be written

$$C = \sum_{n=1}^{N} (s_i - u_n), \tag{1}$$

given that N articles have arrived since the last synchronisation at times u_1 to u_N. We wish to find a synchronisation time which minimises this cost. Unfortunately we don't know when the updates will arrive, we can only estimate the rate of arrival using our probabilistic model, so we need to minimise the *expected cost*.

Consider how the expected cost will depend on three ordered synchronisation times s_{i-1}, s_i and s_{i+1}:

$$\mathcal{C}(s_{i-1}, s_i, s_{i+1}) = \int_{s_{i-1}}^{s_i} \lambda(t)(s_i - t)dt + \int_{s_i}^{s_{i+1}} \lambda(t)(s_{i+1} - t)dt, \tag{2}$$

where $\lambda(t)$ is a time varying function representing the estimated rate at which updates are occuring at the master news database, and is obtained from the probabilistic model. The first term in Equation 2 is the expected cost that will be incurred when we synchronise at time s_i. Note the inclusion of the second term, which is the expected cost that will be incurred when we synchronise at time s_{i+1}. This cost also depends on s_i.

We can now minimise the expected cost with respect to each s_i given the neighbouring synchronisations. This may be done through an iterative algorithm where passes are made through the proposed synchronisation times optimising each one in turn until convergence is achieved.

An alternative to minimising staleness is to maintain the same level of staleness that could be achieved using the uniform approach, but to achieve this using fewer synchronisations per day. This has the benefit of reducing the number of messages required (potentially reducing the financial cost of using the system), and has implications for saving power.

3.1. Results

Figure 1 shows some of the elements of our approach. The histogram based probabilistic model for weekdays for the business part of the news database is shown as boxes on the graph, generated using the updates occurring in May and June 2000. The tool automatically determines the periodicity of the data, and for the business part of the news database this is a weekday and weekend periodicity. Therefore, the weekdays are mapped into one twenty-four hour period and created a histogram to represent that twenty-four hours, and this is shown in Figure 1 (there is a separate model for the weekend which is not shown here). Six synchronisations were requested per day, and the vertical solid lines in Figure 1 show the optimal synchronisation times, to minimise the staleness. The vertical dotted lines in the lower half of the graph identify synchronisation times as taken from a 'uniform' strategy that synchronises every four hours.

Table 1 presents some results for our news database application, showing the staleness achieved when each of the four named databases is replicated individually, and when they are all replicated. It shows the average time in minutes between an article becoming available on the master news database and appearing in the replica on the mobile device, for articles arriving on weekdays in July 2000. The figures show the results when six synchronisations per day were used, with both the uniform and adaptive synchronisation times. The uniform approach started at midnight, as is shown in Figure 1[3]. The percentage decrease in staleness for adaptive over uniform is shown. In the final column the number of synchronisations required by the adaptive approach to achieve a similar average staleness of articles as the uniform approach is given, with the observed average staleness shown in brackets afterwards.

[3]It should be noted the effect of starting the uniform synchronisation at other times does not impact the results significantly.

Classification	Staleness (mins)		% Decrease in staleness for adaptive over uniform	Number of synchronisations for equivalent staleness
	Uniform	Adaptive		
Business	123.3	87.9	29%	4 (130.2)
Entertainment	113.7	78.6	31%	4 (119.4)
Health	131.8	94.6	28%	5 (125.4)
UK	120.2	109.5	9%	5 (127.2)
All	122.3	105.2	14%	5 (132.9)

Table 1. Results for weekdays of the month of July 2000 using six synchronisations and comparing uniform with optimised synchronisation times, together with the number of optimised synchronisations required to achieve comparable levels of staleness as six uniform synchronisations.

4. Example Mobile Email Client

We now demonstrate the use of our approach in a second example, a mobile email client. A central server is being used to store email, and the mobile email client running on a mobile device synchronises with the central email server. We assume that a pull model is used, so the mobile email reader initiates the synchronisation. The mobile email client is similar to the mobile news database, and uses the probabilistic models to indicate to the user the likely divergence between the email cached on the mobile device, and the to control when synchronisations should occur. These are calculated using a similar cost function to that used in the News Database example.

A tool was used to create a log of when email arrived (email folders updated) for six Microsoft Exchange users over the months of January and February 2001, by using information held in the Exchange server. The update log for January was used to create the probabilistic models and the information for February was used to evaluate the performance of the synchronisation times chosen. The probabilistic models were created automatically, with the tool calculating the periodicity of the data being modelled. For the six users, four were modelled using a weekly periodicity, and the other two were modelled using a weekday/weekend periodicity.

Figure 2 presents results for the optimally chosen synchronisation times for the six Exchange users, showing the mean percentage decrease in staleness versus the number of synchronisations per day, with the error bars representing +/- one standard deviation. For the uniform synchronisation, all possible synchronisation times (based on a five minute granularity) were tried. So, for 24 synchronisations per day, the scenarios tried included a synchronisation occurred on every hour, then 5 minutes past every hour, then 10 minutes past every hour, etc. In this example, 11 sets of synchronisation times would be calculated and the average staleness was evaluated, and used to represent the staleness for the uniform approach.

The number of synchronisations was varied between 1 and 24 synchronisations per day. The results show clearly

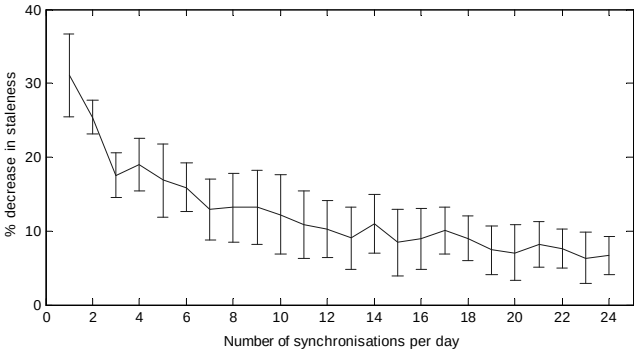

Figure 2. Reduction in staleness of email items for between 1 and 24 synchronisations per day for six users.

that regardless of the number of synchronisations per day the average staleness of email is reduced.

5. Other Applications and Future Work

Web Cache Web caches have become an integral part of the World Wide Web. Caches are embedded within web browsers as well as throughout the general infrastructure of the web. Their role is to replicate web pages thereby reducing latency of web page access, bandwidth usage and web server load. The HTTP protocol [4] provides support for web caches, allowing the life times of object received to be explicitly set, and for fields providing explicit instructions to caches on how to treat a particular page. A number of schemes have been proposed to allow caches to work more efficiently [6].

Many web sites are automatically generated using tools that could generate logs of when the pages are updated. These logs could then be used by our tools to generate the probabilistic models of each page. The models are small (approximately 100 bytes) and can be carried within the HTTP protocol from the web server which generates the web page to web caches and browsers. Enabled web caches and browsers can then use these models to make decisions

about when a cached page is still acceptable (under user specified parameters), and inform a user the likelihood that the page has been updated.

Calendar The examples used so far involve data that cannot be modified at the replica. Whilst interesting, clearly the most exciting applications are those that allow the all replicas to be modified. Therefore, we have been looking at a calendar application, where a single user's calendar is replicated, and there are multiple people concurrently accessing and updating the calendar (for example a manager and their secretary).

As with the mobile news database and mobile email reader, the calendar application can calculate synchronisation times. More interestingly, the user interface can use the information to adapt, for example, indicate appointment slots that are less likely to lead to conflicts when synchronisation occurs. Also, potentially, the models can be used to provide *just-in-time update propagation*. Imagine a scenario where a secretary has access to a salesperson's calendar. The salesperson and the secretary are the only people who make appointments and the secretary works only weekdays. If on a Saturday the salesperson makes an appointment for the following week this need not be propagated until Monday morning, when the secretary arrives. However, if on a Tuesday morning the salesperson makes an appointment for the next day this should be propagated immediately because the secretary will be booking appoints on the same day. If the same salesperson also makes an appointment on the Tuesday morning for a month in the future, this might not need to be propagated immediately because, for example, the secretary never makes appointments more than a week in advance. Using the models of how the data evolves, the write update's propagation can be delayed until the system thinks that by delaying any longer the chance of conflict increases significantly. Furthermore, the updates can be propagated in any order. Thus the advantages of delaying propagation are that it may be possible to package the updates in packets more efficiently, saving cost and bandwidth, as well as the potential to delay until a cheaper communication medium becomes available. We are currently working on evaluating the feasibility of *just-in-time update propagation*.

6. Related work

Cho et al. [3, 2] examine techniques for a web crawler to maintain a large repository of web pages. Their work is focused on when each of the web pages should be checked in order to maintain a fresh and consistent repository. This involves estimating the rate of update of web pages, which is assumed to be constant. The main aim of their approach is to obtain an estimate of the rate of page update given that they haven't observed every update of the page. Due to its nature, a web crawler is unable to obtain a complete log of page updates and as a result it is non-trivial to obtain an unbiased estimate of the rate of change of the page. The model they prescribe is too simple, however, to enable decisions on the granularity of a day about when synchronisations should be made.

In contrast, we assume that we have complete logs of updates. Our models can be much richer than an estimation of a constant rate and thus provide more information for decision making. We are also making the information available to the application, allowing it to choose when to synchronise (picking an optimal time to synchronise rather than just picking the order in which to synchronise elements in the database), and also allowing the application to generally alter its behaviour based on the expected divergence.

In TACT [7] a number of metrics are proposed that allow the control of the replica divergence: *Numerical error*, *Order error* and *Staleness*. However, these metrics control the divergence rather than attempt to estimate its probable divergence.

7. Conclusions

This paper has described how probabilistic models can be used to estimate replica divergence and has given examples as to the sort of decisions that can be made based upon these models to improve the end-user experience. We have given a proof-of-concept demonstration of the approach in two simple applications and have suggested further, more complex examples to which the methods can be applied.

References

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] J. Cho and H. Garcia-Molina. Estimating frequency of change. Submitted for publication., 2000.

[3] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proc. 2000 ACM Int. Conf. on Management of Data (SIGMOD)*, May 2000.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Http version 1.1 rfc 2616. http://www.w3.org/Protocols/Specs.html, 1999.

[5] K. V. Mardia and P. E. Jupp. *Directional Statistics*. John Wiley and Sons, Chichester, West Sussex, 2nd edition, 1999.

[6] H. Yu, L. Breslau, and S. Shenker. A scalable web cache consistency architecture. In *Proc. ACM SIGCOMM'99*, Boston, MA, USA, September 1999.

[7] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *4th Symp. on Operating System Design and Implementation (OSDI)*, Oct 2000.